

Run-Time Library (RTL) :
Reference guide.

Free Pascal version 3.2.4-rc2:
Reference guide for RTL units.
Document version 3.2.4
September 2026

Michaël Van Canneyt

Contents

0.1	Overview	154
1	Reference for unit 'BaseUnix'	155
1.1	Used units	155
1.2	Overview	155
1.3	Constants, types and variables	155
1.3.1	Constants	155
1.3.2	Types	180
1.4	Procedures and functions	193
1.4.1	CreateShellArgV	193
1.4.2	FpAccess	193
1.4.3	FpAlarm	194
1.4.4	FpChdir	195
1.4.5	FpChmod	195
1.4.6	FpChown	196
1.4.7	FpClose	197
1.4.8	FpClosedir	198
1.4.9	FpDup	198
1.4.10	FpDup2	199
1.4.11	FpExecv	200
1.4.12	FpExecve	201
1.4.13	FpExit	202
1.4.14	FpFcntl	202
1.4.15	fpgdfillset	203
1.4.16	fpFD_CLR	203
1.4.17	fpFD_ISSET	203
1.4.18	fpFD_SET	203
1.4.19	fpFD_ZERO	204
1.4.20	FpFork	204
1.4.21	FPFStat	204
1.4.22	FpFtruncate	205

1.4.23	FpGetcwd	206
1.4.24	FpGetegid	206
1.4.25	FpGetEnv	206
1.4.26	fpgeterno	207
1.4.27	FpGeteuid	207
1.4.28	FpGetgid	208
1.4.29	FpGetgroups	208
1.4.30	FpGetpggrp	208
1.4.31	FpGetpid	209
1.4.32	FpGetppid	209
1.4.33	fpGetPriority	210
1.4.34	FpGetRLimit	210
1.4.35	FpGetsid	210
1.4.36	FpGetuid	211
1.4.37	FpIOCtl	211
1.4.38	FpKill	212
1.4.39	FpLink	212
1.4.40	FpLseek	213
1.4.41	fpLstat	214
1.4.42	FpMkdir	215
1.4.43	FpMkfifo	216
1.4.44	Fp mmap	216
1.4.45	Fp mprotect	217
1.4.46	Fp munmap	218
1.4.47	FpNanoSleep	218
1.4.48	fpNice	219
1.4.49	FpOpen	220
1.4.50	FpOpendir	221
1.4.51	FpPause	222
1.4.52	FpPipe	222
1.4.53	FpPoll	223
1.4.54	FppRead	223
1.4.55	FppWrite	224
1.4.56	FpRead	224
1.4.57	FpReaddir	225
1.4.58	fpReadLink	225
1.4.59	FpReadV	227
1.4.60	FpRename	227
1.4.61	FpRmdir	228
1.4.62	fpSelect	228

1.4.63	fpseterno	229
1.4.64	FpSetgid	230
1.4.65	fpSetPriority	230
1.4.66	FpSetRLimit	230
1.4.67	FpSetsid	231
1.4.68	fpsettimeofday	231
1.4.69	FpSetuid	231
1.4.70	FPSigaction	232
1.4.71	FpSigAddSet	233
1.4.72	FpSigDelSet	233
1.4.73	FpsigEmptySet	233
1.4.74	FpSigFillSet	234
1.4.75	FpSigIsMember	234
1.4.76	FpSignal	234
1.4.77	FpSigPending	235
1.4.78	FpSigProcMask	235
1.4.79	FpSigSuspend	236
1.4.80	FpSigTimedWait	236
1.4.81	FpSleep	236
1.4.82	FpStat	237
1.4.83	fpSymlink	238
1.4.84	fpS_ISBLK	239
1.4.85	fpS_ISCHR	239
1.4.86	fpS_ISDIR	240
1.4.87	fpS_ISFIFO	240
1.4.88	fpS_ISLNK	240
1.4.89	fpS_ISREG	241
1.4.90	fpS_ISSOCK	241
1.4.91	fpTime	241
1.4.92	FpTimes	242
1.4.93	FpUmask	242
1.4.94	FpUname	243
1.4.95	FpUnlink	243
1.4.96	FpUtime	243
1.4.97	FpWait	244
1.4.98	FpWaitPid	245
1.4.99	FpWrite	245
1.4.100	FpWriteV	246
1.4.101	FreeShellArgV	246
1.4.102	wexitStatus	246

1.4.103	wifexited	246
1.4.104	wifsignaled	247
1.4.105	wstopsig	247
1.4.106	wtermSIG	247
1.5	Dir	247
1.6	Dirent	248
1.7	FLock	248
1.8	iovec	248
1.9	pollfd	248
1.10	sigactionrec	249
1.11	Stat	249
1.12	tfpreg	249
1.13	tfpstate	250
1.14	tfpxreg	250
1.15	tfpx_sw_bytes	251
1.16	timezone	251
1.17	tms	251
1.18	TRLimit	251
1.19	tsigaltstack	252
1.20	TSigContext	252
1.21	tsiginfo	253
1.22	TUcontext	254
1.23	txmmreg	254
1.24	UTimBuf	254
1.25	UtsName	255
2	Reference for unit 'Character'	256
2.1	Used units	256
2.2	Overview	256
2.3	Constants, types and variables	256
2.3.1	Types	256
2.4	Procedures and functions	258
2.4.1	ConvertFromUtf32	258
2.4.2	ConvertToUtf32	259
2.4.3	GetNumericValue	259
2.4.4	GetUnicodeCategory	259
2.4.5	IsControl	260
2.4.6	IsDigit	260
2.4.7	IsHighSurrogate	260
2.4.8	IsLetter	260

2.4.9	IsLetterOrDigit	261
2.4.10	IsLower	261
2.4.11	IsLowSurrogate	261
2.4.12	IsNumber	261
2.4.13	IsPunctuation	262
2.4.14	IsSeparator	262
2.4.15	IsSurrogate	262
2.4.16	IsSurrogatePair	262
2.4.17	IsSymbol	263
2.4.18	IsUpper	263
2.4.19	IsWhiteSpace	263
2.4.20	ToLower	263
2.4.21	ToUpper	264
2.5	TCharacter	264
2.5.1	Description	264
2.5.2	Method overview	264
2.5.3	TCharacter.Create	264
2.5.4	TCharacter.ConvertFromUtf32	265
2.5.5	TCharacter.ConvertToUtf32	265
2.5.6	TCharacter.GetNumericValue	266
2.5.7	TCharacter.GetUnicodeCategory	266
2.5.8	TCharacter.IsControl	266
2.5.9	TCharacter.IsDigit	267
2.5.10	TCharacter.IsSurrogate	267
2.5.11	TCharacter.IsHighSurrogate	268
2.5.12	TCharacter.IsLowSurrogate	268
2.5.13	TCharacter.IsSurrogatePair	269
2.5.14	TCharacter.IsLetter	269
2.5.15	TCharacter.IsLetterOrDigit	269
2.5.16	TCharacter.IsLower	270
2.5.17	TCharacter.IsNumber	270
2.5.18	TCharacter.IsPunctuation	271
2.5.19	TCharacter.IsSeparator	271
2.5.20	TCharacter.IsSymbol	272
2.5.21	TCharacter.IsUpper	272
2.5.22	TCharacter.IsWhiteSpace	272
2.5.23	TCharacter.ToLower	273
2.5.24	TCharacter.ToUpper	273

3 Reference for unit 'charset'

275

3.1	Used units	275
3.2	Overview	275
3.3	Constants, types and variables	275
3.3.1	Constants	275
3.3.2	Types	275
3.4	Procedures and functions	276
3.4.1	getascii	276
3.4.2	getmap	277
3.4.3	getunicode	277
3.4.4	loadbinaryunicodemapping	277
3.4.5	loadunicodemapping	278
3.4.6	mappingavailable	278
3.4.7	registerbinarymapping	278
3.4.8	registermapping	279
3.5	reversecharmapping	279
3.6	TSerializedMapHeader	279
3.7	tunicodecharmapping	279
3.8	tunicodemap	280
4	Reference for unit 'Classes'	281
4.1	Used units	281
4.2	Overview	281
4.3	Constants, types and variables	281
4.3.1	Constants	281
4.3.2	Types	284
4.3.3	Variables	302
4.4	Procedures and functions	303
4.4.1	ActivateClassGroup	303
4.4.2	BeginGlobalLoading	304
4.4.3	BinToHex	304
4.4.4	Bounds	304
4.4.5	CheckSynchronize	304
4.4.6	ClassGroupOf	305
4.4.7	CollectionsEqual	305
4.4.8	EndGlobalLoading	305
4.4.9	ExtractStrings	305
4.4.10	FindClass	306
4.4.11	FindGlobalComponent	306
4.4.12	FindIdentToInt	306
4.4.13	FindIntToIdent	307

4.4.14	FindNestedComponent	307
4.4.15	GetClass	307
4.4.16	GetFixupInstanceNames	307
4.4.17	GetFixupReferenceNames	308
4.4.18	GlobalFixupReferences	308
4.4.19	GroupDescendentsWith	308
4.4.20	HexToBin	308
4.4.21	IdentToInt	309
4.4.22	IfThen	309
4.4.23	InitComponentRes	309
4.4.24	InitInheritedComponent	309
4.4.25	IntToIdent	310
4.4.26	InvalidPoint	310
4.4.27	LineStart	310
4.4.28	NotifyGlobalLoading	310
4.4.29	ObjectBinaryToText	311
4.4.30	ObjectResourceToText	311
4.4.31	ObjectTextToBinary	311
4.4.32	ObjectTextToResource	311
4.4.33	Point	312
4.4.34	PointsEqual	312
4.4.35	ReadComponentRes	312
4.4.36	ReadComponentResEx	312
4.4.37	ReadComponentResFile	313
4.4.38	Rect	313
4.4.39	RedirectFixupReferences	313
4.4.40	RegisterClass	313
4.4.41	RegisterClassAlias	314
4.4.42	RegisterClasses	314
4.4.43	RegisterComponents	314
4.4.44	RegisterFindGlobalComponentProc	314
4.4.45	RegisterInitComponentHandler	315
4.4.46	RegisterIntegerConsts	315
4.4.47	RegisterNoIcon	315
4.4.48	RegisterNonActiveX	316
4.4.49	RemoveFixupReferences	316
4.4.50	RemoveFixups	316
4.4.51	SmallPoint	316
4.4.52	StartClassGroup	317
4.4.53	UnRegisterClass	317

4.4.54	UnRegisterClasses	317
4.4.55	UnregisterFindGlobalComponentProc	317
4.4.56	UnRegisterModuleClasses	318
4.4.57	WriteComponentResFile	318
4.5	TIdentMapEntry	318
4.6	TStringItem	318
4.7	EBitsError	319
4.7.1	Description	319
4.8	EClassNotFound	319
4.8.1	Description	319
4.9	EComponentError	319
4.9.1	Description	319
4.10	EFCreateError	319
4.10.1	Description	319
4.11	EFilerError	319
4.11.1	Description	319
4.12	EOpenError	320
4.12.1	Description	320
4.13	EInvalidImage	320
4.13.1	Description	320
4.14	EInvalidOperation	320
4.14.1	Description	320
4.15	EMethodNotFound	320
4.15.1	Description	320
4.16	EObserver	320
4.16.1	Description	320
4.17	EOutOfResources	320
4.17.1	Description	320
4.18	EParserError	321
4.18.1	Description	321
4.19	EReadError	321
4.19.1	Description	321
4.20	EResNotFound	321
4.20.1	Description	321
4.21	EStreamError	321
4.21.1	Description	321
4.22	EStringListError	322
4.22.1	Description	322
4.23	EThread	322
4.23.1	Description	322

4.24	EThreadDestroyCalled	322
4.24.1	Description	322
4.25	EThreadExternalException	322
4.25.1	Description	322
4.26	EWriteError	322
4.26.1	Description	322
4.27	IDesignerNotify	323
4.27.1	Description	323
4.27.2	Method overview	323
4.27.3	IDesignerNotify.Modified	323
4.27.4	IDesignerNotify.Notification	323
4.28	IFPObserved	323
4.28.1	Description	323
4.28.2	Method overview	324
4.28.3	IFPObserved.FPOAttachObserver	324
4.28.4	IFPObserved.FPODetachObserver	324
4.28.5	IFPObserved.FPONotifyObservers	324
4.29	IFPObserver	325
4.29.1	Description	325
4.29.2	Method overview	325
4.29.3	IFPObserver.FPOObservedChanged	325
4.30	IInterfaceComponentReference	325
4.30.1	Description	325
4.30.2	Method overview	326
4.30.3	IInterfaceComponentReference.GetComponent	326
4.31	IInterfaceList	326
4.31.1	Description	326
4.31.2	Method overview	326
4.31.3	Property overview	327
4.31.4	IInterfaceList.Get	327
4.31.5	IInterfaceList.GetCapacity	327
4.31.6	IInterfaceList.GetCount	327
4.31.7	IInterfaceList.Put	327
4.31.8	IInterfaceList.SetCapacity	328
4.31.9	IInterfaceList.SetCount	328
4.31.10	IInterfaceList.Clear	328
4.31.11	IInterfaceList.Delete	328
4.31.12	IInterfaceList.Exchange	329
4.31.13	IInterfaceList.First	329
4.31.14	IInterfaceList.IndexOf	329

4.31.15	InterfaceList.Add	329
4.31.16	InterfaceList.Insert	329
4.31.17	InterfaceList.Last	330
4.31.18	InterfaceList.Remove	330
4.31.19	InterfaceList.Lock	330
4.31.20	InterfaceList.Unlock	330
4.31.21	InterfaceList.Capacity	330
4.31.22	InterfaceList.Count	331
4.31.23	InterfaceList.Items	331
4.32	InterfaceListEx	331
4.32.1	Method overview	331
4.32.2	InterfaceListEx.GetEnumerator	331
4.33	IStreamPersist	331
4.33.1	Description	331
4.33.2	Method overview	332
4.33.3	IStreamPersist.LoadFromStream	332
4.33.4	IStreamPersist.SaveToStream	332
4.34	IStringsAdapter	332
4.34.1	Description	332
4.34.2	Method overview	332
4.34.3	IStringsAdapter.ReferenceStrings	332
4.34.4	IStringsAdapter.ReleaseStrings	333
4.35	IVCLComObject	333
4.35.1	Description	333
4.35.2	Method overview	333
4.35.3	IVCLComObject.GetTypeInfoCount	333
4.35.4	IVCLComObject.GetTypeInfo	333
4.35.5	IVCLComObject.GetIDsOfNames	334
4.35.6	IVCLComObject.Invoke	334
4.35.7	IVCLComObject.SafeCallException	334
4.35.8	IVCLComObject.FreeOnRelease	335
4.36	TAbstractObjectReader	335
4.36.1	Description	335
4.36.2	Method overview	336
4.36.3	TAbstractObjectReader.FlushBuffer	336
4.36.4	TAbstractObjectReader.NextValue	336
4.36.5	TAbstractObjectReader.ReadValue	337
4.36.6	TAbstractObjectReader.BeginRootComponent	337
4.36.7	TAbstractObjectReader.BeginComponent	337
4.36.8	TAbstractObjectReader.BeginProperty	337

4.36.9	TAbstractObjectReader.Read	338
4.36.10	TAbstractObjectReader.ReadBinary	338
4.36.11	TAbstractObjectReader.ReadFloat	338
4.36.12	TAbstractObjectReader.ReadSingle	338
4.36.13	TAbstractObjectReader.ReadDouble	339
4.36.14	TAbstractObjectReader.ReadDate	339
4.36.15	TAbstractObjectReader.ReadCurrency	339
4.36.16	TAbstractObjectReader.ReadIdent	339
4.36.17	TAbstractObjectReader.ReadInt8	340
4.36.18	TAbstractObjectReader.ReadInt16	340
4.36.19	TAbstractObjectReader.ReadInt32	341
4.36.20	TAbstractObjectReader.ReadInt64	341
4.36.21	TAbstractObjectReader.ReadSet	341
4.36.22	TAbstractObjectReader.ReadSignature	342
4.36.23	TAbstractObjectReader.ReadStr	342
4.36.24	TAbstractObjectReader.ReadString	342
4.36.25	TAbstractObjectReader.ReadWideString	343
4.36.26	TAbstractObjectReader.ReadUnicodeString	343
4.36.27	TAbstractObjectReader.SkipComponent	343
4.36.28	TAbstractObjectReader.SkipValue	343
4.37	TAbstractObjectWriter	344
4.37.1	Description	344
4.37.2	Method overview	344
4.37.3	TAbstractObjectWriter.BeginCollection	344
4.37.4	TAbstractObjectWriter.BeginComponent	344
4.37.5	TAbstractObjectWriter.WriteSignature	345
4.37.6	TAbstractObjectWriter.BeginList	345
4.37.7	TAbstractObjectWriter.EndList	345
4.37.8	TAbstractObjectWriter.BeginProperty	345
4.37.9	TAbstractObjectWriter.EndProperty	345
4.37.10	TAbstractObjectWriter.FlushBuffer	346
4.37.11	TAbstractObjectWriter.Write	346
4.37.12	TAbstractObjectWriter.WriteBinary	346
4.37.13	TAbstractObjectWriter.WriteBoolean	346
4.37.14	TAbstractObjectWriter.WriteFloat	346
4.37.15	TAbstractObjectWriter.WriteSingle	347
4.37.16	TAbstractObjectWriter.WriteDate	347
4.37.17	TAbstractObjectWriter.WriteCurrency	347
4.37.18	TAbstractObjectWriter.WriteIdent	347
4.37.19	TAbstractObjectWriter.WriteInteger	347

4.37.20	TAbstractObjectWriter.WriteUInt64	348
4.37.21	TAbstractObjectWriter.WriteVariant	348
4.37.22	TAbstractObjectWriter.WriteMethodName	348
4.37.23	TAbstractObjectWriter.WriteSet	348
4.37.24	TAbstractObjectWriter.WriteString	348
4.37.25	TAbstractObjectWriter.WriteWideString	349
4.37.26	TAbstractObjectWriter.WriteUnicodeString	349
4.38	TBasicAction	349
4.38.1	Description	349
4.38.2	Method overview	349
4.38.3	Property overview	350
4.38.4	TBasicAction.Create	350
4.38.5	TBasicAction.Destroy	350
4.38.6	TBasicAction.HandlesTarget	350
4.38.7	TBasicAction.UpdateTarget	351
4.38.8	TBasicAction.ExecuteTarget	351
4.38.9	TBasicAction.Execute	351
4.38.10	TBasicAction.RegisterChanges	351
4.38.11	TBasicAction.UnRegisterChanges	352
4.38.12	TBasicAction.Update	352
4.38.13	TBasicAction.Suspended	352
4.38.14	TBasicAction.ActionComponent	352
4.38.15	TBasicAction.OnExecute	353
4.38.16	TBasicAction.OnUpdate	353
4.38.17	TBasicAction.Clients	353
4.39	TBasicActionLink	353
4.39.1	Description	353
4.39.2	Method overview	354
4.39.3	Property overview	354
4.39.4	TBasicActionLink.Create	354
4.39.5	TBasicActionLink.Destroy	354
4.39.6	TBasicActionLink.Execute	354
4.39.7	TBasicActionLink.Update	355
4.39.8	TBasicActionLink.Action	355
4.39.9	TBasicActionLink.OnChange	355
4.40	TBinaryObjectReader	356
4.40.1	Description	356
4.40.2	Method overview	356
4.40.3	TBinaryObjectReader.Create	356
4.40.4	TBinaryObjectReader.Destroy	357

4.40.5	TBinaryObjectReader.NextValue	357
4.40.6	TBinaryObjectReader.ReadValue	357
4.40.7	TBinaryObjectReader.BeginRootComponent	357
4.40.8	TBinaryObjectReader.BeginComponent	358
4.40.9	TBinaryObjectReader.BeginProperty	358
4.40.10	TBinaryObjectReader.Read	358
4.40.11	TBinaryObjectReader.ReadBinary	358
4.40.12	TBinaryObjectReader.ReadFloat	358
4.40.13	TBinaryObjectReader.ReadSingle	359
4.40.14	TBinaryObjectReader.ReadDouble	359
4.40.15	TBinaryObjectReader.ReadDate	359
4.40.16	TBinaryObjectReader.ReadCurrency	359
4.40.17	TBinaryObjectReader.ReadIdent	359
4.40.18	TBinaryObjectReader.ReadInt8	360
4.40.19	TBinaryObjectReader.ReadInt16	360
4.40.20	TBinaryObjectReader.ReadInt32	360
4.40.21	TBinaryObjectReader.ReadInt64	360
4.40.22	TBinaryObjectReader.ReadSet	361
4.40.23	TBinaryObjectReader.ReadSignature	361
4.40.24	TBinaryObjectReader.ReadStr	361
4.40.25	TBinaryObjectReader.ReadString	361
4.40.26	TBinaryObjectReader.ReadWideString	362
4.40.27	TBinaryObjectReader.ReadUnicodeString	362
4.40.28	TBinaryObjectReader.SkipComponent	362
4.40.29	TBinaryObjectReader.SkipValue	362
4.41	TBinaryObjectWriter	362
4.41.1	Description	362
4.41.2	Method overview	363
4.41.3	Property overview	363
4.41.4	TBinaryObjectWriter.Create	363
4.41.5	TBinaryObjectWriter.Destroy	363
4.41.6	TBinaryObjectWriter.WriteSignature	364
4.41.7	TBinaryObjectWriter.FlushBuffer	364
4.41.8	TBinaryObjectWriter.BeginCollection	364
4.41.9	TBinaryObjectWriter.BeginComponent	364
4.41.10	TBinaryObjectWriter.BeginList	364
4.41.11	TBinaryObjectWriter.EndList	364
4.41.12	TBinaryObjectWriter.BeginProperty	365
4.41.13	TBinaryObjectWriter.EndProperty	365
4.41.14	TBinaryObjectWriter.Write	365

4.41.15	TBinaryObjectWriter.WriteBinary	365
4.41.16	TBinaryObjectWriter.WriteBoolean	365
4.41.17	TBinaryObjectWriter.WriteFloat	365
4.41.18	TBinaryObjectWriter.WriteSingle	366
4.41.19	TBinaryObjectWriter.WriteDate	366
4.41.20	TBinaryObjectWriter.WriteCurrency	366
4.41.21	TBinaryObjectWriter.WriteIdent	366
4.41.22	TBinaryObjectWriter.WriteInteger	366
4.41.23	TBinaryObjectWriter.WriteUInt64	366
4.41.24	TBinaryObjectWriter.WriteMethodName	367
4.41.25	TBinaryObjectWriter.WriteSet	367
4.41.26	TBinaryObjectWriter.WriteString	367
4.41.27	TBinaryObjectWriter.WriteString	367
4.41.28	TBinaryObjectWriter.WriteString	367
4.41.29	TBinaryObjectWriter.WriteString	367
4.41.30	TBinaryObjectWriter.WriteString	367
4.41.31	TBinaryObjectWriter.Version	368
4.42	TBits	368
4.42.1	Description	368
4.42.2	Method overview	368
4.42.3	Property overview	369
4.42.4	TBits.Create	369
4.42.5	TBits.Destroy	369
4.42.6	TBits.GetFSize	369
4.42.7	TBits.SetOn	370
4.42.8	TBits.Clear	370
4.42.9	TBits.Clearall	370
4.42.10	TBits.CopyBits	370
4.42.11	TBits.AndBits	371
4.42.12	TBits.OrBits	371
4.42.13	TBits.XorBits	371
4.42.14	TBits.NotBits	372
4.42.15	TBits.Get	372
4.42.16	TBits.Grow	372
4.42.17	TBits.Equals	372
4.42.18	TBits.SetIndex	373
4.42.19	TBits.FindFirstBit	373
4.42.20	TBits.FindNextBit	373
4.42.21	TBits.FindPrevBit	374
4.42.22	TBits.OpenBit	374

4.42.23	TBits.Bits	374
4.42.24	TBits.Size	375
4.43	TBytesStream	375
4.43.1	Description	375
4.43.2	Method overview	375
4.43.3	Property overview	375
4.43.4	TBytesStream.Create	375
4.43.5	TBytesStream.Bytes	375
4.44	TCollection	376
4.44.1	Description	376
4.44.2	Method overview	376
4.44.3	Property overview	376
4.44.4	TCollection.Create	376
4.44.5	TCollection.Destroy	377
4.44.6	TCollection.Owner	377
4.44.7	TCollection.Add	377
4.44.8	TCollection.Assign	377
4.44.9	TCollection.BeginUpdate	378
4.44.10	TCollection.Clear	378
4.44.11	TCollection.EndUpdate	379
4.44.12	TCollection.Delete	379
4.44.13	TCollection.GetEnumerator	379
4.44.14	TCollection.GetNamePath	379
4.44.15	TCollection.Insert	380
4.44.16	TCollection.FindItemID	380
4.44.17	TCollection.Exchange	380
4.44.18	TCollection.Move	380
4.44.19	TCollection.Sort	381
4.44.20	TCollection.Count	381
4.44.21	TCollection.ItemClass	381
4.44.22	TCollection.Items	381
4.45	TCollectionEnumerator	382
4.45.1	Description	382
4.45.2	Method overview	382
4.45.3	Property overview	382
4.45.4	TCollectionEnumerator.Create	382
4.45.5	TCollectionEnumerator.GetCurrent	382
4.45.6	TCollectionEnumerator.MoveNext	383
4.45.7	TCollectionEnumerator.Current	383
4.46	TCollectionItem	383

4.46.1	Description	383
4.46.2	Method overview	384
4.46.3	Property overview	384
4.46.4	TCollectionItem.Create	384
4.46.5	TCollectionItem.Destroy	384
4.46.6	TCollectionItem.GetNamePath	384
4.46.7	TCollectionItem.Collection	385
4.46.8	TCollectionItem.ID	385
4.46.9	TCollectionItem.Index	385
4.46.10	TCollectionItem.DisplayName	386
4.47	TComponent	386
4.47.1	Description	386
4.47.2	Interfaces overview	386
4.47.3	Method overview	387
4.47.4	Property overview	387
4.47.5	TComponent.Notification	387
4.47.6	TComponent.WriteState	388
4.47.7	TComponent.Create	388
4.47.8	TComponent.Destroy	388
4.47.9	TComponent.BeforeDestruction	389
4.47.10	TComponent.DestroyComponents	389
4.47.11	TComponent.Destroying	389
4.47.12	TComponent.ExecuteAction	389
4.47.13	TComponent.FindComponent	390
4.47.14	TComponent.FreeNotification	390
4.47.15	TComponent.RemoveFreeNotification	390
4.47.16	TComponent.FreeOnRelease	390
4.47.17	TComponent.GetEnumerator	390
4.47.18	TComponent.GetNamePath	391
4.47.19	TComponent.GetParentComponent	391
4.47.20	TComponent.HasParent	391
4.47.21	TComponent.InsertComponent	391
4.47.22	TComponent.RemoveComponent	392
4.47.23	TComponent.SafeCallException	392
4.47.24	TComponent.SetSubComponent	392
4.47.25	TComponent.UpdateAction	392
4.47.26	TComponent.IsImplementorOf	393
4.47.27	TComponent.ReferenceInterface	393
4.47.28	TComponent.ComObject	393
4.47.29	TComponent.Components	393

4.47.30	TComponent.ComponentCount	394
4.47.31	TComponent.ComponentIndex	394
4.47.32	TComponent.ComponentState	394
4.47.33	TComponent.ComponentStyle	395
4.47.34	TComponent.DesignInfo	395
4.47.35	TComponent.Owner	395
4.47.36	TComponent.VCLComObject	395
4.47.37	TComponent.Name	396
4.47.38	TComponent.Tag	396
4.48	TComponentEnumerator	396
4.48.1	Description	396
4.48.2	Method overview	396
4.48.3	Property overview	397
4.48.4	TComponentEnumerator.Create	397
4.48.5	TComponentEnumerator.GetCurrent	397
4.48.6	TComponentEnumerator.MoveNext	397
4.48.7	TComponentEnumerator.Current	397
4.49	TCustomMemoryStream	398
4.49.1	Description	398
4.49.2	Method overview	398
4.49.3	Property overview	398
4.49.4	TCustomMemoryStream.Read	398
4.49.5	TCustomMemoryStream.Seek	398
4.49.6	TCustomMemoryStream.SaveToStream	399
4.49.7	TCustomMemoryStream.SaveToFile	399
4.49.8	TCustomMemoryStream.Memory	400
4.50	TDataModule	400
4.50.1	Description	400
4.50.2	Method overview	400
4.50.3	Property overview	400
4.50.4	TDataModule.Create	401
4.50.5	TDataModule.CreateNew	401
4.50.6	TDataModule.Destroy	401
4.50.7	TDataModule.AfterConstruction	401
4.50.8	TDataModule.BeforeDestruction	402
4.50.9	TDataModule.DesignOffset	402
4.50.10	TDataModule.DesignSize	402
4.50.11	TDataModule.DesignPPI	402
4.50.12	TDataModule.OnCreate	403
4.50.13	TDataModule.OnDestroy	403

4.50.14	TDataModule.OldCreateOrder	403
4.51	TFiler	403
4.51.1	Description	403
4.51.2	Method overview	404
4.51.3	Property overview	404
4.51.4	TFiler.DefineProperty	404
4.51.5	TFiler.DefineBinaryProperty	404
4.51.6	TFiler.FlushBuffer	404
4.51.7	TFiler.Root	405
4.51.8	TFiler.LookupRoot	405
4.51.9	TFiler.Ancestor	405
4.51.10	TFiler.IgnoreChildren	405
4.52	TFileStream	406
4.52.1	Description	406
4.52.2	Method overview	406
4.52.3	Property overview	406
4.52.4	TFileStream.Create	406
4.52.5	TFileStream.Destroy	407
4.52.6	TFileStream.Flush	407
4.52.7	TFileStream.FileName	407
4.53	TFPList	408
4.53.1	Description	408
4.53.2	Method overview	408
4.53.3	Property overview	408
4.53.4	TFPList.Destroy	409
4.53.5	TFPList.AddList	409
4.53.6	TFPList.Add	409
4.53.7	TFPList.Clear	409
4.53.8	TFPList.Delete	410
4.53.9	TFPList.Error	410
4.53.10	TFPList.Exchange	410
4.53.11	TFPList.Expand	410
4.53.12	TFPList.Extract	411
4.53.13	TFPList.First	411
4.53.14	TFPList.GetEnumerator	411
4.53.15	TFPList.Contains	411
4.53.16	TFPList.IndexOf	411
4.53.17	TFPList.IndexOfItem	412
4.53.18	TFPList.Insert	412
4.53.19	TFPList.Last	412

4.53.20	TFPList.Move	412
4.53.21	TFPList.Assign	413
4.53.22	TFPList.Remove	413
4.53.23	TFPList.Pack	413
4.53.24	TFPList.Sort	413
4.53.25	TFPList.ForEachCall	414
4.53.26	TFPList.Capacity	414
4.53.27	TFPList.Count	414
4.53.28	TFPList.Items	415
4.53.29	TFPList.List	415
4.54	TFPListEnumerator	415
4.54.1	Description	415
4.54.2	Method overview	415
4.54.3	Property overview	415
4.54.4	TFPListEnumerator.Create	415
4.54.5	TFPListEnumerator.GetCurrent	416
4.54.6	TFPListEnumerator.MoveNext	416
4.54.7	TFPListEnumerator.Current	416
4.55	THandleStream	417
4.55.1	Description	417
4.55.2	Method overview	417
4.55.3	Property overview	417
4.55.4	THandleStream.Create	417
4.55.5	THandleStream.Read	417
4.55.6	THandleStream.Write	418
4.55.7	THandleStream.Seek	418
4.55.8	THandleStream.Handle	418
4.56	TInterfacedPersistent	419
4.56.1	Description	419
4.56.2	Interfaces overview	419
4.56.3	Method overview	419
4.56.4	TInterfacedPersistent.QueryInterface	419
4.56.5	TInterfacedPersistent.AfterConstruction	419
4.57	TInterfaceList	419
4.57.1	Description	419
4.57.2	Interfaces overview	420
4.57.3	Method overview	420
4.57.4	Property overview	420
4.57.5	TInterfaceList.Create	420
4.57.6	TInterfaceList.Destroy	420

4.57.7	TInterfaceList.Clear	421
4.57.8	TInterfaceList.Delete	421
4.57.9	TInterfaceList.Exchange	421
4.57.10	TInterfaceList.First	421
4.57.11	TInterfaceList.GetEnumerator	422
4.57.12	TInterfaceList.IndexOf	422
4.57.13	TInterfaceList.Add	422
4.57.14	TInterfaceList.Insert	422
4.57.15	TInterfaceList.Last	423
4.57.16	TInterfaceList.Remove	423
4.57.17	TInterfaceList.Lock	423
4.57.18	TInterfaceList.Unlock	423
4.57.19	TInterfaceList.Expand	424
4.57.20	TInterfaceList.Capacity	424
4.57.21	TInterfaceList.Count	424
4.57.22	TInterfaceList.Items	424
4.58	TInterfaceListEnumerator	425
4.58.1	Description	425
4.58.2	Method overview	425
4.58.3	Property overview	425
4.58.4	TInterfaceListEnumerator.Create	425
4.58.5	TInterfaceListEnumerator.GetCurrent	425
4.58.6	TInterfaceListEnumerator.MoveNext	426
4.58.7	TInterfaceListEnumerator.Current	426
4.59	TList	426
4.59.1	Description	426
4.59.2	Interfaces overview	426
4.59.3	Method overview	427
4.59.4	Property overview	427
4.59.5	TList.Create	427
4.59.6	TList.Destroy	427
4.59.7	TList.FPOAttachObserver	428
4.59.8	TList.FPODetachObserver	428
4.59.9	TList.FPONotifyObservers	428
4.59.10	TList.AddList	429
4.59.11	TList.Add	429
4.59.12	TList.Clear	429
4.59.13	TList.Delete	429
4.59.14	TList.Error	430
4.59.15	TList.Exchange	430

4.59.16	TList.Expand	430
4.59.17	TList.Extract	430
4.59.18	TList.First	431
4.59.19	TList.GetEnumerator	431
4.59.20	TList.Contains	431
4.59.21	TList.IndexOf	431
4.59.22	TList.Insert	431
4.59.23	TList.Last	432
4.59.24	TList.Move	432
4.59.25	TList.Assign	432
4.59.26	TList.Remove	432
4.59.27	TList.Pack	433
4.59.28	TList.Sort	433
4.59.29	TList.Capacity	433
4.59.30	TList.Count	434
4.59.31	TList.Items	434
4.59.32	TList.List	434
4.60	TListEnumerator	434
4.60.1	Description	434
4.60.2	Method overview	434
4.60.3	Property overview	435
4.60.4	TListEnumerator.Create	435
4.60.5	TListEnumerator.GetCurrent	435
4.60.6	TListEnumerator.MoveNext	435
4.60.7	TListEnumerator.Current	435
4.61	TMemoryStream	436
4.61.1	Description	436
4.61.2	Method overview	436
4.61.3	TMemoryStream.Destroy	436
4.61.4	TMemoryStream.Clear	436
4.61.5	TMemoryStream.LoadFromStream	436
4.61.6	TMemoryStream.LoadFromFile	437
4.61.7	TMemoryStream.SetSize	437
4.61.8	TMemoryStream.Write	437
4.62	TOwnedCollection	438
4.62.1	Description	438
4.62.2	Method overview	438
4.62.3	TOwnedCollection.Create	438
4.63	TOwnerStream	438
4.63.1	Description	438

4.63.2	Method overview	439
4.63.3	Property overview	439
4.63.4	TOwnerStream.Create	439
4.63.5	TOwnerStream.Destroy	439
4.63.6	TOwnerStream.Source	439
4.63.7	TOwnerStream.SourceOwner	440
4.64	TParser	440
4.64.1	Description	440
4.64.2	Method overview	440
4.64.3	Property overview	441
4.64.4	TParser.Create	441
4.64.5	TParser.Destroy	441
4.64.6	TParser.CheckToken	441
4.64.7	TParser.CheckTokenSymbol	441
4.64.8	TParser.Error	442
4.64.9	TParser.ErrorFmt	442
4.64.10	TParser.ErrorStr	442
4.64.11	TParser.HexToBinary	442
4.64.12	TParser.NextToken	443
4.64.13	TParser.SourcePos	443
4.64.14	TParser.TokenComponentIdent	443
4.64.15	TParser.TokenFloat	444
4.64.16	TParser.TokenInt	444
4.64.17	TParser.TokenString	444
4.64.18	TParser.TokenWideString	445
4.64.19	TParser.TokenSymbolIs	445
4.64.20	TParser.FloatType	445
4.64.21	TParser.SourceLine	446
4.64.22	TParser.Token	446
4.65	TPersistent	446
4.65.1	Description	446
4.65.2	Interfaces overview	447
4.65.3	Method overview	447
4.65.4	TPersistent.Destroy	447
4.65.5	TPersistent.Assign	447
4.65.6	TPersistent.FPOAttachObserver	448
4.65.7	TPersistent.FPODetachObserver	448
4.65.8	TPersistent.FPONotifyObservers	448
4.65.9	TPersistent.GetNamePath	448
4.66	TProxyStream	449

4.66.1	Description	449
4.66.2	Method overview	449
4.66.3	TProxyStream.Create	449
4.66.4	TProxyStream.Read	449
4.66.5	TProxyStream.Write	449
4.66.6	TProxyStream.Seek	449
4.66.7	TProxyStream.Check	450
4.67	TRawByteStringStream	450
4.67.1	Description	450
4.67.2	Method overview	450
4.67.3	TRawByteStringStream.Create	450
4.67.4	TRawByteStringStream.DataString	450
4.67.5	TRawByteStringStream.ReadString	451
4.67.6	TRawByteStringStream.WriteString	451
4.68	TReader	451
4.68.1	Description	451
4.68.2	Method overview	452
4.68.3	Property overview	453
4.68.4	TReader.Create	453
4.68.5	TReader.Destroy	453
4.68.6	TReader.FlushBuffer	453
4.68.7	TReader.BeginReferences	454
4.68.8	TReader.CheckValue	454
4.68.9	TReader.DefineProperty	454
4.68.10	TReader.DefineBinaryProperty	454
4.68.11	TReader.EndOfList	454
4.68.12	TReader.EndReferences	455
4.68.13	TReader.FixupReferences	455
4.68.14	TReader.SkipValue	455
4.68.15	TReader.NextValue	455
4.68.16	TReader.Read	455
4.68.17	TReader.ReadBoolean	455
4.68.18	TReader.ReadChar	456
4.68.19	TReader.ReadWideChar	456
4.68.20	TReader.ReadUnicodeChar	456
4.68.21	TReader.ReadCollection	456
4.68.22	TReader.ReadComponent	456
4.68.23	TReader.ReadComponents	457
4.68.24	TReader.ReadFloat	457
4.68.25	TReader.ReadSingle	457

4.68.26	TReader.ReadDouble	457
4.68.27	TReader.ReadDate	457
4.68.28	TReader.ReadCurrency	457
4.68.29	TReader.ReadIdent	458
4.68.30	TReader.ReadInteger	458
4.68.31	TReader.ReadInt64	458
4.68.32	TReader.ReadSet	458
4.68.33	TReader.ReadListBegin	458
4.68.34	TReader.ReadListEnd	459
4.68.35	TReader.ReadRootComponent	459
4.68.36	TReader.ReadVariant	459
4.68.37	TReader.ReadSignature	459
4.68.38	TReader.ReadString	459
4.68.39	TReader.ReadWideString	460
4.68.40	TReader.ReadUnicodeString	460
4.68.41	TReader.ReadValue	460
4.68.42	TReader.CopyValue	460
4.68.43	TReader.Driver	460
4.68.44	TReader.Owner	461
4.68.45	TReader.Parent	461
4.68.46	TReader.OnError	461
4.68.47	TReader.OnPropertyNotFound	461
4.68.48	TReader.OnFindMethod	462
4.68.49	TReader.OnSetMethodProperty	462
4.68.50	TReader.OnSetName	462
4.68.51	TReader.OnReferenceName	462
4.68.52	TReader.OnAncestorNotFound	462
4.68.53	TReader.OnCreateComponent	463
4.68.54	TReader.OnFindComponentClass	463
4.68.55	TReader.OnReadStringProperty	463
4.69	TRecall	463
4.69.1	Description	463
4.69.2	Method overview	464
4.69.3	Property overview	464
4.69.4	TRecall.Create	464
4.69.5	TRecall.Destroy	464
4.69.6	TRecall.Store	464
4.69.7	TRecall.Forget	465
4.69.8	TRecall.Reference	465
4.70	TResourceStream	465

4.70.1	Description	465
4.70.2	Method overview	465
4.70.3	TResourceStream.Create	466
4.70.4	TResourceStream.CreateFromID	466
4.70.5	TResourceStream.Destroy	466
4.71	TStream	466
4.71.1	Description	466
4.71.2	Method overview	467
4.71.3	Property overview	467
4.71.4	TStream.Read	467
4.71.5	TStream.Write	468
4.71.6	TStream.Seek	468
4.71.7	TStream.ReadBuffer	469
4.71.8	TStream.WriteBuffer	469
4.71.9	TStream.CopyFrom	470
4.71.10	TStream.ReadComponent	470
4.71.11	TStream.ReadComponentRes	470
4.71.12	TStream.WriteComponent	471
4.71.13	TStream.WriteComponentRes	471
4.71.14	TStream.WriteDescendent	471
4.71.15	TStream.WriteDescendentRes	471
4.71.16	TStream.WriteResourceHeader	472
4.71.17	TStream.FixupResourceHeader	472
4.71.18	TStream.ReadResHeader	472
4.71.19	TStream.ReadByte	472
4.71.20	TStream.ReadWord	473
4.71.21	TStream.ReadDWord	473
4.71.22	TStream.ReadQWord	473
4.71.23	TStream.ReadAnsiString	474
4.71.24	TStream.WriteByte	474
4.71.25	TStream.WriteWord	474
4.71.26	TStream.WriteDWord	474
4.71.27	TStream.WriteQWord	475
4.71.28	TStream.WriteAnsiString	475
4.71.29	TStream.Position	475
4.71.30	TStream.Size	476
4.72	TStreamAdapter	476
4.72.1	Description	476
4.72.2	Interfaces overview	476
4.72.3	Method overview	476

4.72.4	Property overview	477
4.72.5	TStreamAdapter.Create	477
4.72.6	TStreamAdapter.Destroy	477
4.72.7	TStreamAdapter.Read	477
4.72.8	TStreamAdapter.Write	477
4.72.9	TStreamAdapter.Seek	478
4.72.10	TStreamAdapter.SetSize	478
4.72.11	TStreamAdapter.CopyTo	478
4.72.12	TStreamAdapter.Commit	479
4.72.13	TStreamAdapter.Revert	479
4.72.14	TStreamAdapter.LockRegion	479
4.72.15	TStreamAdapter.UnlockRegion	479
4.72.16	TStreamAdapter.Stat	480
4.72.17	TStreamAdapter.Clone	480
4.72.18	TStreamAdapter.Stream	480
4.72.19	TStreamAdapter.StreamOwnership	480
4.73	TStringList	481
4.73.1	Description	481
4.73.2	Method overview	481
4.73.3	Property overview	481
4.73.4	TStringList.Create	481
4.73.5	TStringList.Destroy	482
4.73.6	TStringList.Add	482
4.73.7	TStringList.AddObject	482
4.73.8	TStringList.Clear	482
4.73.9	TStringList.Delete	482
4.73.10	TStringList.Exchange	483
4.73.11	TStringList.Find	483
4.73.12	TStringList.IndexOf	483
4.73.13	TStringList.Insert	483
4.73.14	TStringList.Sort	484
4.73.15	TStringList.CustomSort	484
4.73.16	TStringList.Duplicates	484
4.73.17	TStringList.Sorted	485
4.73.18	TStringList.CaseSensitive	485
4.73.19	TStringList.OnChange	485
4.73.20	TStringList.OnChanging	486
4.73.21	TStringList.OwnsObjects	486
4.73.22	TStringList.SortStyle	486
4.74	TStrings	486

4.74.1	Description	486
4.74.2	Method overview	488
4.74.3	Property overview	489
4.74.4	TStrings.Create	489
4.74.5	TStrings.Destroy	490
4.74.6	TStrings.ToObjectArray	490
4.74.7	TStrings.ToStringArray	490
4.74.8	TStrings.Add	491
4.74.9	TStrings.AddObject	491
4.74.10	TStrings.AddPair	491
4.74.11	TStrings.AddStrings	492
4.74.12	TStrings.SetStrings	492
4.74.13	TStrings.AddText	492
4.74.14	TStrings.AddCommaText	492
4.74.15	TStrings.AddDelimitedtext	493
4.74.16	TStrings.Append	493
4.74.17	TStrings.Assign	493
4.74.18	TStrings.BeginUpdate	494
4.74.19	TStrings.Clear	494
4.74.20	TStrings.Delete	494
4.74.21	TStrings.EndUpdate	495
4.74.22	TStrings.Equals	495
4.74.23	TStrings.Exchange	495
4.74.24	TStrings.ExtractName	496
4.74.25	TStrings.Filter	496
4.74.26	TStrings.Fill	496
4.74.27	TStrings.ForEach	496
4.74.28	TStrings.GetEnumerator	497
4.74.29	TStrings.GetNameValue	497
4.74.30	TStrings.GetText	497
4.74.31	TStrings.Contains	497
4.74.32	TStrings.ContainsObject	497
4.74.33	TStrings.IndexOf	497
4.74.34	TStrings.ContainsName	498
4.74.35	TStrings.IndexOfName	498
4.74.36	TStrings.IndexOfObject	498
4.74.37	TStrings.Insert	499
4.74.38	TStrings.InsertObject	499
4.74.39	TStrings.LastIndexOf	499
4.74.40	TStrings.LoadFromFile	500

4.74.41	TStrings.LoadFromStream	500
4.74.42	TStrings.Map	500
4.74.43	TStrings.Move	501
4.74.44	TStrings.Pop	501
4.74.45	TStrings.Reduce	501
4.74.46	TStrings.Reverse	502
4.74.47	TStrings.SaveToFile	502
4.74.48	TStrings.SaveToStream	502
4.74.49	TStrings.Shift	503
4.74.50	TStrings.Slice	503
4.74.51	TStrings.SetText	503
4.74.52	TStrings.AlwaysQuote	504
4.74.53	TStrings.Capacity	504
4.74.54	TStrings.CommaText	504
4.74.55	TStrings.Count	505
4.74.56	TStrings.DefaultEncoding	506
4.74.57	TStrings.DelimitedText	506
4.74.58	TStrings.Delimiter	506
4.74.59	TStrings.Encoding	507
4.74.60	TStrings.LineBreak	507
4.74.61	TStrings.MissingNameValueSeparatorAction	507
4.74.62	TStrings.Names	507
4.74.63	TStrings.NameValueSeparator	508
4.74.64	TStrings.Objects	508
4.74.65	TStrings.Options	508
4.74.66	TStrings.QuoteChar	509
4.74.67	TStrings.SkipLastLineBreak	509
4.74.68	TStrings.TrailingLineBreak	509
4.74.69	TStrings.StrictDelimiter	509
4.74.70	TStrings.Strings	510
4.74.71	TStrings.StringsAdapter	510
4.74.72	TStrings.Text	510
4.74.73	TStrings.TextLineBreakStyle	511
4.74.74	TStrings.UseLocale	511
4.74.75	TStrings.ValueFromIndex	512
4.74.76	TStrings.Values	512
4.74.77	TStrings.WriteBOM	512
4.75	TStringsEnumerator	513
4.75.1	Description	513
4.75.2	Method overview	513

4.75.3	Property overview	513
4.75.4	TStringsEnumerator.Create	513
4.75.5	TStringsEnumerator.GetCurrent	513
4.75.6	TStringsEnumerator.MoveNext	513
4.75.7	TStringsEnumerator.Current	514
4.76	TStringStream	514
4.76.1	Description	514
4.76.2	Method overview	514
4.76.3	Property overview	515
4.76.4	TStringStream.Create	515
4.76.5	TStringStream.CreateRaw	515
4.76.6	TStringStream.Destroy	515
4.76.7	TStringStream.ReadUnicodeString	516
4.76.8	TStringStream.WriteUnicodeString	516
4.76.9	TStringStream.ReadAnsiString	516
4.76.10	TStringStream.WriteAnsiString	516
4.76.11	TStringStream.ReadString	517
4.76.12	TStringStream.WriteString	517
4.76.13	TStringStream.DataString	517
4.76.14	TStringStream.UnicodeDataString	517
4.76.15	TStringStream.OwnsEncoding	518
4.76.16	TStringStream.Encoding	518
4.77	TTextObjectWriter	518
4.77.1	Description	518
4.78	TThread	518
4.78.1	Description	518
4.78.2	Method overview	519
4.78.3	Property overview	519
4.78.4	TThread.Execute	520
4.78.5	TThread.Synchronize	520
4.78.6	TThread.Queue	520
4.78.7	TThread.ForceQueue	520
4.78.8	TThread.Create	521
4.78.9	TThread.Destroy	521
4.78.10	TThread.CreateAnonymousThread	521
4.78.11	TThread.NameThreadForDebugging	521
4.78.12	TThread.SetReturnValue	522
4.78.13	TThread.CheckTerminated	522
4.78.14	TThread.RemoveQueuedEvents	522
4.78.15	TThread.SpinWait	523

4.78.16	TThread.Sleep	523
4.78.17	TThread.Yield	523
4.78.18	TThread.GetSystemTimes	523
4.78.19	TThread.GetCPUUsage	524
4.78.20	TThread.GetTickCount	524
4.78.21	TThread.GetTickCount64	524
4.78.22	TThread.ExecuteInThread	524
4.78.23	TThread.AfterConstruction	529
4.78.24	TThread.Start	529
4.78.25	TThread.Resume	529
4.78.26	TThread.Suspend	530
4.78.27	TThread.Terminate	530
4.78.28	TThread.WaitFor	530
4.78.29	TThread.CurrentIsMain	530
4.78.30	TThread.CurrentThread	530
4.78.31	TThread.Current	531
4.78.32	TThread.ProcessorCount	531
4.78.33	TThread.IsSingleProcessor	531
4.78.34	TThread.FreeOnTerminate	532
4.78.35	TThread.Handle	532
4.78.36	TThread.ExternalThread	532
4.78.37	TThread.Priority	533
4.78.38	TThread.Suspended	533
4.78.39	TThread.Finished	533
4.78.40	TThread.ThreadID	533
4.78.41	TThread.OnTerminate	534
4.78.42	TThread.FatalException	534
4.79	TThreadList	534
4.79.1	Description	534
4.79.2	Method overview	534
4.79.3	Property overview	534
4.79.4	TThreadList.Create	535
4.79.5	TThreadList.Destroy	535
4.79.6	TThreadList.Add	535
4.79.7	TThreadList.Clear	535
4.79.8	TThreadList.LockList	536
4.79.9	TThreadList.Remove	536
4.79.10	TThreadList.UnlockList	536
4.79.11	TThreadList.Duplicates	536
4.80	TWriter	537

4.80.1	Description	537
4.80.2	Method overview	537
4.80.3	Property overview	538
4.80.4	TWriter.Create	538
4.80.5	TWriter.Destroy	538
4.80.6	TWriter.FlushBuffer	538
4.80.7	TWriter.DefineProperty	538
4.80.8	TWriter.DefineBinaryProperty	539
4.80.9	TWriter.Write	539
4.80.10	TWriter.WriteBoolean	539
4.80.11	TWriter.WriteCollection	539
4.80.12	TWriter.WriteComponent	539
4.80.13	TWriter.WriteChar	540
4.80.14	TWriter.WriteWideChar	540
4.80.15	TWriter.WriteDescendent	540
4.80.16	TWriter.WriteFloat	540
4.80.17	TWriter.WriteSingle	540
4.80.18	TWriter.WriteDate	541
4.80.19	TWriter.WriteCurrency	541
4.80.20	TWriter.WriteIdent	541
4.80.21	TWriter.WriteInteger	541
4.80.22	TWriter.WriteSet	541
4.80.23	TWriter.WriteListBegin	542
4.80.24	TWriter.WriteListEnd	542
4.80.25	TWriter.WriteSignature	542
4.80.26	TWriter.WriteRootComponent	542
4.80.27	TWriter.WriteString	542
4.80.28	TWriter.WriteWideString	543
4.80.29	TWriter.WriteUnicodeString	543
4.80.30	TWriter.WriteVariant	543
4.80.31	TWriter.RootAncestor	543
4.80.32	TWriter.OnFindAncestor	543
4.80.33	TWriter.OnWriteMethodProperty	544
4.80.34	TWriter.OnWriteStringProperty	544
4.80.35	TWriter.Driver	544
4.80.36	TWriter.PropertyPath	544
5	Reference for unit 'clocale'	545
5.1	Used units	545
5.2	Overview	545

6	Reference for unit 'cmem'	546
6.1	Used units	546
6.2	Overview	546
6.3	Constants, types and variables	546
6.3.1	Constants	546
6.4	Procedures and functions	546
6.4.1	CAlloc	546
6.4.2	Free	547
6.4.3	Malloc	547
6.4.4	ReAlloc	547
7	Reference for unit 'collation_de'	548
7.1	Used units	548
7.2	Overview	548
8	Reference for unit 'collation_es'	549
8.1	Used units	549
8.2	Overview	549
9	Reference for unit 'collation_fr_ca'	550
9.1	Used units	550
9.2	Overview	550
10	Reference for unit 'collation_ja'	551
10.1	Used units	551
10.2	Overview	551
11	Reference for unit 'collation_ko'	552
11.1	Used units	552
11.2	Overview	552
12	Reference for unit 'collation_ru'	553
12.1	Used units	553
12.2	Overview	553
13	Reference for unit 'collation_sv'	554
13.1	Used units	554
13.2	Overview	554
14	Reference for unit 'collation_zh'	555
14.1	Used units	555
14.2	Overview	555

15 Reference for unit 'cp1250'	556
15.1 Used units	556
15.2 Overview	556
16 Reference for unit 'cp1251'	557
16.1 Used units	557
16.2 Overview	557
17 Reference for unit 'cp1252'	558
17.1 Used units	558
17.2 Overview	558
18 Reference for unit 'cp1253'	559
18.1 Used units	559
18.2 Overview	559
19 Reference for unit 'cp1254'	560
19.1 Used units	560
19.2 Overview	560
20 Reference for unit 'cp1255'	561
20.1 Used units	561
20.2 Overview	561
21 Reference for unit 'cp1256'	562
21.1 Used units	562
21.2 Overview	562
22 Reference for unit 'cp1257'	563
22.1 Used units	563
22.2 Overview	563
23 Reference for unit 'cp1258'	564
23.1 Used units	564
23.2 Overview	564
24 Reference for unit 'cp437'	565
24.1 Used units	565
24.2 Overview	565
25 Reference for unit 'cp646'	566
25.1 Used units	566
25.2 Overview	566

26 Reference for unit 'cp850'	567
26.1 Used units	567
26.2 Overview	567
27 Reference for unit 'cp852'	568
27.1 Used units	568
27.2 Overview	568
28 Reference for unit 'cp856'	569
28.1 Used units	569
28.2 Overview	569
29 Reference for unit 'cp866'	570
29.1 Used units	570
29.2 Overview	570
30 Reference for unit 'cp874'	571
30.1 Used units	571
30.2 Overview	571
31 Reference for unit 'cp8859_1'	572
31.1 Used units	572
31.2 Overview	572
32 Reference for unit 'cp8859_2'	573
32.1 Used units	573
32.2 Overview	573
33 Reference for unit 'cp8859_5'	574
33.1 Used units	574
33.2 Overview	574
34 Reference for unit 'cp895'	575
34.1 Used units	575
34.2 Overview	575
35 Reference for unit 'cp932'	576
35.1 Used units	576
35.2 Overview	576
36 Reference for unit 'cp936'	577
36.1 Used units	577
36.2 Overview	577

37 Reference for unit 'cp949'	578
37.1 Used units	578
37.2 Overview	578
38 Reference for unit 'cp950'	579
38.1 Used units	579
38.2 Overview	579
39 Reference for unit 'cpall'	580
39.1 Used units	581
39.2 Overview	582
40 Reference for unit 'Crt'	583
40.1 Used units	583
40.2 Overview	583
40.3 Constants, types and variables	584
40.3.1 Constants	584
40.3.2 Types	586
40.3.3 Variables	586
40.4 Procedures and functions	587
40.4.1 AssignCrt	587
40.4.2 ClrEol	588
40.4.3 ClrScr	589
40.4.4 cursorbig	589
40.4.5 cursoroff	589
40.4.6 cursoron	590
40.4.7 Delay	590
40.4.8 DelLine	590
40.4.9 GotoXY	591
40.4.10 HighVideo	592
40.4.11 InsLine	592
40.4.12 KeyPressed	593
40.4.13 LowVideo	593
40.4.14 NormVideo	594
40.4.15 NoSound	594
40.4.16 ReadKey	594
40.4.17 Sound	595
40.4.18 TextBackground	595
40.4.19 TextColor	596
40.4.20 TextMode	597
40.4.21 WhereX	597

40.4.22	WhereY	597
40.4.23	Window	598
40.5	TCharAttr	598
41	Reference for unit 'cthreads'	600
41.1	Used units	600
41.2	Overview	600
41.3	Procedures and functions	601
41.3.1	SetCThreadManager	601
42	Reference for unit 'ctypes'	602
42.1	Used units	602
42.2	Overview	602
42.3	Constants, types and variables	602
42.3.1	Types	602
43	Reference for unit 'cwstring'	608
43.1	Used units	608
43.2	Overview	608
43.3	Procedures and functions	608
43.3.1	SetCWidestringManager	608
44	Reference for unit 'DateUtils'	610
44.1	Used units	610
44.2	Overview	610
44.3	Constants, types and variables	610
44.3.1	Constants	610
44.3.2	Types	613
44.4	Procedures and functions	613
44.4.1	CompareDate	613
44.4.2	CompareDateTime	614
44.4.3	CompareTime	615
44.4.4	DateInRange	616
44.4.5	DateOf	617
44.4.6	DateTimeInRange	617
44.4.7	DateTimeToDosDateTime	618
44.4.8	DateTimeToJulianDate	618
44.4.9	DateTimeToMac	618
44.4.10	DateTimeToModifiedJulianDate	619
44.4.11	DateTimeToUnix	619
44.4.12	DateToISO8601	619

44.4.13	DayOf	620
44.4.14	DayOfTheMonth	620
44.4.15	DayOfTheWeek	620
44.4.16	DayOfTheYear	621
44.4.17	DaysBetween	621
44.4.18	DaysInAMonth	622
44.4.19	DaysInAYear	623
44.4.20	DaysInMonth	623
44.4.21	DaysInYear	624
44.4.22	DaySpan	624
44.4.23	DecodeDateDay	625
44.4.24	DecodeDateMonthWeek	626
44.4.25	DecodeDateTime	626
44.4.26	DecodeDateWeek	627
44.4.27	DecodeDayOfWeekInMonth	628
44.4.28	DosDateTimeToDateTime	628
44.4.29	EncodeDateDay	628
44.4.30	EncodeDateMonthWeek	629
44.4.31	EncodeDateTime	629
44.4.32	EncodeDateWeek	630
44.4.33	EncodeDayOfWeekInMonth	630
44.4.34	EncodeTimeInterval	630
44.4.35	EndOfADay	631
44.4.36	EndOfAMonth	631
44.4.37	EndOfAWeek	632
44.4.38	EndOfAYear	633
44.4.39	EndOfTheDay	633
44.4.40	EndOfTheMonth	634
44.4.41	EndOfTheWeek	634
44.4.42	EndOfTheYear	635
44.4.43	HourOf	635
44.4.44	HourOfTheDay	635
44.4.45	HourOfTheMonth	636
44.4.46	HourOfTheWeek	636
44.4.47	HourOfTheYear	637
44.4.48	HoursBetween	637
44.4.49	HourSpan	638
44.4.50	IncDay	639
44.4.51	IncHour	639
44.4.52	IncMilliSecond	640

44.4.53	IncMinute	640
44.4.54	IncSecond	641
44.4.55	IncWeek	641
44.4.56	IncYear	642
44.4.57	InvalidDateDayError	642
44.4.58	InvalidDateMonthWeekError	642
44.4.59	InvalidDateTimeError	643
44.4.60	InvalidDateWeekError	643
44.4.61	InvalidDayOfWeekInMonthError	644
44.4.62	IsAM	644
44.4.63	IsInLeapYear	644
44.4.64	ISO8601ToDate	644
44.4.65	ISO8601ToDateDef	645
44.4.66	IsPM	646
44.4.67	IsSameDay	646
44.4.68	IsSameMonth	647
44.4.69	IsToday	647
44.4.70	IsValidDate	648
44.4.71	IsValidDateDay	648
44.4.72	IsValidDateMonthWeek	649
44.4.73	IsValidDateTime	650
44.4.74	IsValidDateWeek	651
44.4.75	IsValidTime	652
44.4.76	JulianDateToDateTime	652
44.4.77	LocalTimeToUniversal	652
44.4.78	MacTimeStampToUnix	652
44.4.79	MacToDateTime	653
44.4.80	MilliSecondOf	653
44.4.81	MilliSecondOfTheDay	653
44.4.82	MilliSecondOfTheHour	653
44.4.83	MilliSecondOfTheMinute	654
44.4.84	MilliSecondOfTheMonth	654
44.4.85	MilliSecondOfTheSecond	654
44.4.86	MilliSecondOfTheWeek	655
44.4.87	MilliSecondOfTheYear	655
44.4.88	MilliSecondsBetween	656
44.4.89	MilliSecondSpan	656
44.4.90	MinuteOf	657
44.4.91	MinuteOfTheDay	657
44.4.92	MinuteOfTheHour	658

44.4.93	MinuteOfTheMonth	658
44.4.94	MinuteOfTheWeek	659
44.4.95	MinuteOfTheYear	659
44.4.96	MinutesBetween	659
44.4.97	MinuteSpan	660
44.4.98	ModifiedJulianDateToDateTime	661
44.4.99	MonthOf	661
44.4.100	MonthOfTheYear	661
44.4.101	MonthsBetween	662
44.4.102	MonthSpan	663
44.4.103	NthDayOfWeek	663
44.4.104	PeriodBetween	664
44.4.105	PreviousDayOfWeek	664
44.4.106	RecodeDate	665
44.4.107	RecodeDateTime	665
44.4.108	RecodeDay	666
44.4.109	RecodeHour	667
44.4.110	RecodeMilliSecond	668
44.4.111	RecodeMinute	668
44.4.112	RecodeMonth	669
44.4.113	RecodeSecond	670
44.4.114	RecodeTime	670
44.4.115	RecodeYear	671
44.4.116	SameDate	672
44.4.117	SameDateTime	672
44.4.118	SameTime	673
44.4.119	ScanDateTime	674
44.4.120	SecondOf	675
44.4.121	SecondOfTheDay	675
44.4.122	SecondOfTheHour	675
44.4.123	SecondOfTheMinute	675
44.4.124	SecondOfTheMonth	676
44.4.125	SecondOfTheWeek	676
44.4.126	SecondOfTheYear	677
44.4.127	SecondsBetween	677
44.4.128	SecondSpan	678
44.4.129	StartOfADay	679
44.4.130	StartOfAMonth	679
44.4.131	StartOfAWeek	680
44.4.132	StartOfAYear	681

44.4.133	StartOfDay	681
44.4.134	StartOfMonth	682
44.4.135	StartOfWeek	682
44.4.136	StartOfYear	683
44.4.137	TimeInRange	683
44.4.138	TimeOf	683
44.4.139	Today	684
44.4.140	Tomorrow	684
44.4.141	TryEncodeDateDay	685
44.4.142	TryEncodeDateMonthWeek	685
44.4.143	TryEncodeDateTime	686
44.4.144	TryEncodeDateWeek	687
44.4.145	TryEncodeDayOfWeekInMonth	687
44.4.146	TryEncodeTimeInterval	688
44.4.147	TryISO8601ToDate	688
44.4.148	TryISOStrToDate	689
44.4.149	TryISOStrToDateTime	690
44.4.150	TryISOStrToTime	691
44.4.151	TryISOTZStrToTZOffset	692
44.4.152	TryJulianDateToDateTime	693
44.4.153	TryModifiedJulianDateToDateTime	693
44.4.154	TryRecodeDateTime	693
44.4.155	UniversalTimeToLocal	694
44.4.156	UnixTimeStampToMac	694
44.4.157	UnixToDateTime	695
44.4.158	WeekOf	695
44.4.159	WeekOfMonth	695
44.4.160	WeekOfYear	696
44.4.161	WeeksBetween	697
44.4.162	WeeksInAYear	697
44.4.163	WeeksInYear	698
44.4.164	WeekSpan	699
44.4.165	WithinPastDays	699
44.4.166	WithinPastHours	700
44.4.167	WithinPastMilliseconds	701
44.4.168	WithinPastMinutes	702
44.4.169	WithinPastMonths	703
44.4.170	WithinPastSeconds	704
44.4.171	WithinPastWeeks	705
44.4.172	WithinPastYears	706

44.4.173	YearOf	707
44.4.174	YearsBetween	708
44.4.175	YearSpan	709
44.4.176	Yesterday	710
44.5	EDatetimeException	712
44.6	ELocalTimeInvalid	712
44.7	TDateTimeHelper	712
44.7.1	Method overview	712
44.7.2	Property overview	713
44.7.3	TDateTimeHelper.Create	713
44.7.4	TDateTimeHelper.CreateLocal	713
44.7.5	TDateTimeHelper.CreateUnixTime	713
44.7.6	TDateTimeHelper.CreateTotalSeconds	713
44.7.7	TDateTimeHelper.GetDate	714
44.7.8	TDateTimeHelper.GetTime	714
44.7.9	TDateTimeHelper.ToString	714
44.7.10	TDateTimeHelper.Format	714
44.7.11	TDateTimeHelper.StartOfYear	714
44.7.12	TDateTimeHelper.EndOfYear	714
44.7.13	TDateTimeHelper.StartOfMonth	714
44.7.14	TDateTimeHelper.EndOfMonth	714
44.7.15	TDateTimeHelper.StartOfWeek	715
44.7.16	TDateTimeHelper.EndOfWeek	715
44.7.17	TDateTimeHelper.StartOfDay	715
44.7.18	TDateTimeHelper.EndOfDay	715
44.7.19	TDateTimeHelper.AddYears	715
44.7.20	TDateTimeHelper.AddMonths	715
44.7.21	TDateTimeHelper.AddDays	715
44.7.22	TDateTimeHelper.AddHours	715
44.7.23	TDateTimeHelper.AddMinutes	715
44.7.24	TDateTimeHelper.AddSeconds	716
44.7.25	TDateTimeHelper.AddMilliseconds	716
44.7.26	TDateTimeHelper.CompareTo	716
44.7.27	TDateTimeHelper.Equals	716
44.7.28	TDateTimeHelper.IsSameDay	716
44.7.29	TDateTimeHelper.InRange	716
44.7.30	TDateTimeHelper.IsInLeapYear	716
44.7.31	TDateTimeHelper.IsToday	716
44.7.32	TDateTimeHelper.IsAM	717
44.7.33	TDateTimeHelper.IsPM	717

44.7.34	TDateTimeHelper.YearsBetween	717
44.7.35	TDateTimeHelper.MonthsBetween	717
44.7.36	TDateTimeHelper.WeeksBetween	717
44.7.37	TDateTimeHelper.DaysBetween	717
44.7.38	TDateTimeHelper.HoursBetween	717
44.7.39	TDateTimeHelper.MinutesBetween	717
44.7.40	TDateTimeHelper.SecondsBetween	717
44.7.41	TDateTimeHelper.MilliSecondsBetween	718
44.7.42	TDateTimeHelper.WithinPastYears	718
44.7.43	TDateTimeHelper.WithinPastMonths	718
44.7.44	TDateTimeHelper.WithinPastWeeks	718
44.7.45	TDateTimeHelper.WithinPastDays	718
44.7.46	TDateTimeHelper.WithinPastHours	718
44.7.47	TDateTimeHelper.WithinPastMinutes	718
44.7.48	TDateTimeHelper.WithinPastSeconds	718
44.7.49	TDateTimeHelper.WithinPastMilliSeconds	719
44.7.50	TDateTimeHelper.WithinYears	719
44.7.51	TDateTimeHelper.WithinMonths	719
44.7.52	TDateTimeHelper.WithinWeeks	719
44.7.53	TDateTimeHelper.WithinDays	719
44.7.54	TDateTimeHelper.WithinHours	719
44.7.55	TDateTimeHelper.WithinMinutes	719
44.7.56	TDateTimeHelper.WithinSeconds	719
44.7.57	TDateTimeHelper.WithinMilliseconds	720
44.7.58	TDateTimeHelper.Now	720
44.7.59	TDateTimeHelper.Today	720
44.7.60	TDateTimeHelper.Yesterday	720
44.7.61	TDateTimeHelper.Tomorrow	720
44.7.62	TDateTimeHelper.Date	720
44.7.63	TDateTimeHelper.Time	720
44.7.64	TDateTimeHelper.DayOfWeek	721
44.7.65	TDateTimeHelper.DayOfYear	721
44.7.66	TDateTimeHelper.Year	721
44.7.67	TDateTimeHelper.Month	721
44.7.68	TDateTimeHelper.Day	721
44.7.69	TDateTimeHelper.Hour	721
44.7.70	TDateTimeHelper.Minute	721
44.7.71	TDateTimeHelper.Second	722
44.7.72	TDateTimeHelper.Millisecond	722
44.7.73	TDateTimeHelper.UnixTime	722

44.7.74	TDateTimeHelper.TotalSeconds	722
44.8	TTimeZone	722
44.8.1	Method overview	722
44.8.2	Property overview	723
44.8.3	TTimeZone.Init	723
44.8.4	TTimeZone.Done	723
44.8.5	TTimeZone.GetUtcOffset	723
44.8.6	TTimeZone.ToUniversalTime	723
44.8.7	TTimeZone.ToLocalTime	723
44.8.8	TTimeZone.GetDisplayName	723
44.8.9	TTimeZone.GetAbbreviation	723
44.8.10	TTimeZone.GetLocalTimeType	724
44.8.11	TTimeZone.HasDST	724
44.8.12	TTimeZone.IsStandardTime	724
44.8.13	TTimeZone.IsInvalidTime	724
44.8.14	TTimeZone.IsAmbiguousTime	724
44.8.15	TTimeZone.IsDaylightTime	724
44.8.16	TTimeZone.ID	724
44.8.17	TTimeZone.DisplayName	724
44.8.18	TTimeZone.Abbreviation	725
44.8.19	TTimeZone.UtcOffset	725
44.8.20	TTimeZone.Local	725
45	Reference for unit 'Dos'	726
45.1	Used units	726
45.2	Overview	726
45.3	System information.	726
45.4	Process handling.	727
45.5	Directory and disk handling.	727
45.6	File handling.	728
45.7	File open mode constants.	728
45.8	File attributes.	728
45.9	Constants, types and variables	729
45.9.1	Constants	729
45.9.2	Types	730
45.9.3	Variables	732
45.10	Procedures and functions	732
45.10.1	AddDisk	732
45.10.2	DiskFree	733
45.10.3	DiskSize	734

45.10.4	DosExitCode	734
45.10.5	DosVersion	735
45.10.6	DTToUnixDate	735
45.10.7	EnvCount	736
45.10.8	EnvStr	736
45.10.9	Exec	737
45.10.10	FExpand	737
45.10.11	FindClose	737
45.10.12	FindFirst	738
45.10.13	FindNext	739
45.10.14	FSearch	739
45.10.15	FSplit	740
45.10.16	GetCBreak	740
45.10.17	GetDate	741
45.10.18	GetEnv	741
45.10.19	GetFAttr	742
45.10.20	GetFTime	743
45.10.21	GetIntVec	743
45.10.22	GetLongName	744
45.10.23	GetMsCount	744
45.10.24	GetShortName	744
45.10.25	GetTime	745
45.10.26	GetVerify	745
45.10.27	Intr	746
45.10.28	Keep	746
45.10.29	MSDos	746
45.10.30	PackTime	747
45.10.31	SetCBreak	747
45.10.32	SetDate	748
45.10.33	SetFAttr	748
45.10.34	SetFTime	749
45.10.35	SetIntVec	749
45.10.36	SetTime	749
45.10.37	SetVerify	749
45.10.38	SwapVectors	750
45.10.39	UnixDateToDt	750
45.10.40	UnpackTime	750
45.10.41	weekday	751
45.11	DateTime	751
45.12	SearchRec	751

46 Reference for unit 'dxeload'	753
46.1 Used units	753
46.2 Overview	753
46.3 Procedures and functions	753
46.3.1 dxe_load	753
47 Reference for unit 'dynlibs'	754
47.1 Used units	754
47.2 Overview	754
47.3 Constants, types and variables	754
47.3.1 Constants	754
47.3.2 Types	754
47.4 Procedures and functions	755
47.4.1 FreeLibrary	755
47.4.2 GetLoadErrorStr	755
47.4.3 GetProcAddress	755
47.4.4 GetProcedureAddress	756
47.4.5 LoadLibrary	756
47.4.6 SafeLoadLibrary	756
47.4.7 UnloadLibrary	757
48 Reference for unit 'emu387'	758
48.1 Used units	758
48.2 Overview	758
48.3 Procedures and functions	758
48.3.1 npxsetup	758
49 Reference for unit 'errors'	760
49.1 Used units	760
49.2 Overview	760
49.3 Constants, types and variables	760
49.3.1 Constants	760
49.4 Procedures and functions	761
49.4.1 PError	761
49.4.2 StrError	762
50 Reference for unit 'exeinfo'	763
50.1 Used units	763
50.2 Overview	763
50.3 Constants, types and variables	763
50.3.1 Types	763

50.4	Procedures and functions	763
50.4.1	CloseExeFile	763
50.4.2	FindExeSection	764
50.4.3	GetModuleByAddr	764
50.4.4	OpenExeFile	764
50.4.5	ReadDebugLink	765
50.5	TExeFile	765
51	Reference for unit 'fgl'	766
51.1	Used units	766
51.2	Overview	766
51.3	Constants, types and variables	766
51.3.1	Constants	766
51.3.2	Types	766
51.4	EListError	767
51.4.1	Description	767
51.5	TFPGInterfacedObjectList	767
51.5.1	Description	767
51.5.2	Method overview	767
51.5.3	Property overview	767
51.5.4	TFPGInterfacedObjectList.Create	768
51.5.5	TFPGInterfacedObjectList.Add	768
51.5.6	TFPGInterfacedObjectList.Extract	768
51.5.7	TFPGInterfacedObjectList.GetEnumerator	768
51.5.8	TFPGInterfacedObjectList.IndexOf	769
51.5.9	TFPGInterfacedObjectList.Insert	769
51.5.10	TFPGInterfacedObjectList.Assign	769
51.5.11	TFPGInterfacedObjectList.AddList	769
51.5.12	TFPGInterfacedObjectList.Remove	770
51.5.13	TFPGInterfacedObjectList.Sort	770
51.5.14	TFPGInterfacedObjectList.First	770
51.5.15	TFPGInterfacedObjectList.Last	771
51.5.16	TFPGInterfacedObjectList.Items	771
51.5.17	TFPGInterfacedObjectList.List	771
51.6	TFPGList	771
51.6.1	Description	771
51.6.2	Method overview	772
51.6.3	Property overview	772
51.6.4	TFPGList.Create	772
51.6.5	TFPGList.ItemIsManaged	772

51.6.6	TFPGList.Add	772
51.6.7	TFPGList.Extract	773
51.6.8	TFPGList.GetEnumerator	773
51.6.9	TFPGList.IndexOf	773
51.6.10	TFPGList.Insert	773
51.6.11	TFPGList.Assign	774
51.6.12	TFPGList.AddList	774
51.6.13	TFPGList.Remove	774
51.6.14	TFPGList.Sort	774
51.6.15	TFPGList.First	775
51.6.16	TFPGList.Last	775
51.6.17	TFPGList.Items	775
51.6.18	TFPGList.List	775
51.7	TFPGListEnumerator	776
51.7.1	Description	776
51.7.2	Method overview	776
51.7.3	Property overview	776
51.7.4	TFPGListEnumerator.Create	776
51.7.5	TFPGListEnumerator.MoveNext	776
51.7.6	TFPGListEnumerator.Current	776
51.8	TFPGMap	777
51.8.1	Description	777
51.8.2	Method overview	777
51.8.3	Property overview	777
51.8.4	TFPGMap.Create	777
51.8.5	TFPGMap.Add	778
51.8.6	TFPGMap.Find	778
51.8.7	TFPGMap.TryGetData	778
51.8.8	TFPGMap.AddOrSetData	778
51.8.9	TFPGMap.IndexOf	779
51.8.10	TFPGMap.IndexOfData	779
51.8.11	TFPGMap.InsertKey	779
51.8.12	TFPGMap.InsertKeyData	779
51.8.13	TFPGMap.Remove	780
51.8.14	TFPGMap.Keys	780
51.8.15	TFPGMap.Data	780
51.8.16	TFPGMap.KeyData	781
51.8.17	TFPGMap.OnCompare	781
51.8.18	TFPGMap.OnKeyCompare	781
51.8.19	TFPGMap.OnDataCompare	782

51.9	TFPGMapInterfacedObjectData	782
51.9.1	Description	782
51.9.2	Method overview	782
51.9.3	Property overview	783
51.9.4	TFPGMapInterfacedObjectData.Create	783
51.9.5	TFPGMapInterfacedObjectData.Add	783
51.9.6	TFPGMapInterfacedObjectData.Find	783
51.9.7	TFPGMapInterfacedObjectData.TryGetData	784
51.9.8	TFPGMapInterfacedObjectData.AddOrSetData	784
51.9.9	TFPGMapInterfacedObjectData.IndexOf	784
51.9.10	TFPGMapInterfacedObjectData.IndexOfData	784
51.9.11	TFPGMapInterfacedObjectData.InsertKey	785
51.9.12	TFPGMapInterfacedObjectData.InsertKeyData	785
51.9.13	TFPGMapInterfacedObjectData.Remove	785
51.9.14	TFPGMapInterfacedObjectData.Keys	785
51.9.15	TFPGMapInterfacedObjectData.Data	786
51.9.16	TFPGMapInterfacedObjectData.KeyData	786
51.9.17	TFPGMapInterfacedObjectData.OnCompare	786
51.9.18	TFPGMapInterfacedObjectData.OnKeyCompare	787
51.9.19	TFPGMapInterfacedObjectData.OnDataCompare	787
51.10	TFPGMapObject	788
51.10.1	Description	788
51.10.2	Method overview	788
51.10.3	Property overview	788
51.10.4	TFPGMapObject.Create	788
51.10.5	TFPGMapObject.Add	788
51.10.6	TFPGMapObject.Find	789
51.10.7	TFPGMapObject.TryGetData	789
51.10.8	TFPGMapObject.AddOrSetData	789
51.10.9	TFPGMapObject.IndexOf	790
51.10.10	TFPGMapObject.IndexOfData	790
51.10.11	TFPGMapObject.InsertKey	790
51.10.12	TFPGMapObject.InsertKeyData	790
51.10.13	TFPGMapObject.Remove	791
51.10.14	TFPGMapObject.Keys	791
51.10.15	TFPGMapObject.Data	791
51.10.16	TFPGMapObject.KeyData	791
51.10.17	TFPGMapObject.OnCompare	792
51.10.18	TFPGMapObject.OnKeyCompare	792
51.10.19	TFPGMapObject.OnDataCompare	792

51.11	TFPGObjectList	793
51.11.1	Description	793
51.11.2	Method overview	793
51.11.3	Property overview	793
51.11.4	TFPGObjectList.Create	794
51.11.5	TFPGObjectList.Add	794
51.11.6	TFPGObjectList.Extract	794
51.11.7	TFPGObjectList.GetEnumerator	794
51.11.8	TFPGObjectList.IndexOf	795
51.11.9	TFPGObjectList.Insert	795
51.11.10	TFPGObjectList.AddList	795
51.11.11	TFPGObjectList.Assign	795
51.11.12	TFPGObjectList.Remove	796
51.11.13	TFPGObjectList.Sort	796
51.11.14	TFPGObjectList.First	796
51.11.15	TFPGObjectList.Last	797
51.11.16	TFPGObjectList.Items	797
51.11.17	TFPGObjectList.List	797
51.11.18	TFPGObjectList.FreeObjects	797
51.12	TFPSList	798
51.12.1	Description	798
51.12.2	Method overview	798
51.12.3	Property overview	798
51.12.4	TFPSList.Create	799
51.12.5	TFPSList.Destroy	799
51.12.6	TFPSList.ItemIsManaged	799
51.12.7	TFPSList.Add	799
51.12.8	TFPSList.Clear	800
51.12.9	TFPSList.Delete	800
51.12.10	TFPSList.DeleteRange	800
51.12.11	TFPSList.Error	800
51.12.12	TFPSList.Exchange	801
51.12.13	TFPSList.Expand	801
51.12.14	TFPSList.Extract	801
51.12.15	TFPSList.IndexOf	801
51.12.16	TFPSList.Insert	802
51.12.17	TFPSList.Move	802
51.12.18	TFPSList.Assign	802
51.12.19	TFPSList.AddList	803
51.12.20	TFPSList.Remove	803

51.12.21	TFPSList.Pack	803
51.12.22	TFPSList.Sort	803
51.12.23	TFPSList.Capacity	804
51.12.24	TFPSList.Count	804
51.12.25	TFPSList.Items	804
51.12.26	TFPSList.ItemSize	805
51.12.27	TFPSList.List	805
51.12.28	TFPSList.First	805
51.12.29	TFPSList.Last	805
51.13	TFPSMap	806
51.13.1	Description	806
51.13.2	Method overview	806
51.13.3	Property overview	806
51.13.4	TFPSMap.Create	806
51.13.5	TFPSMap.Add	807
51.13.6	TFPSMap.Find	807
51.13.7	TFPSMap.IndexOf	807
51.13.8	TFPSMap.IndexOfData	808
51.13.9	TFPSMap.Insert	808
51.13.10	TFPSMap.InsertKey	808
51.13.11	TFPSMap.InsertKeyData	808
51.13.12	TFPSMap.Remove	809
51.13.13	TFPSMap.Sort	809
51.13.14	TFPSMap.Duplicates	809
51.13.15	TFPSMap.KeySize	810
51.13.16	TFPSMap.DataSize	810
51.13.17	TFPSMap.Keys	810
51.13.18	TFPSMap.Data	810
51.13.19	TFPSMap.KeyData	811
51.13.20	TFPSMap.Sorted	811
51.13.21	TFPSMap.OnPtrCompare	811
51.13.22	TFPSMap.OnKeyPtrCompare	811
51.13.23	TFPSMap.OnDataPtrCompare	812
52	Reference for unit 'fpwistring'	813
52.1	Used units	813
52.2	Overview	813
52.3	Constants, types and variables	814
52.3.1	Variables	814
52.4	Procedures and functions	814

52.4.1	GetActiveCollation	814
52.4.2	SetActiveCollation	814
53	Reference for unit 'getopts'	815
53.1	Used units	815
53.2	Overview	815
53.3	Constants, types and variables	815
53.3.1	Constants	815
53.3.2	Types	816
53.3.3	Variables	816
53.4	Procedures and functions	816
53.4.1	GetLongOpts	816
53.4.2	GetOpt	817
53.5	TOption	819
53.5.1	Method overview	820
53.5.2	TOption.SetOption	820
54	Reference for unit 'go32'	821
54.1	Used units	821
54.2	Overview	821
54.3	Real mode callbacks.	821
54.4	Executing software interrupts.	822
54.5	Software interrupts.	825
54.6	Hardware interrupts.	825
54.7	Disabling interrupts.	827
54.8	Creating your own interrupt handlers.	827
54.9	Protected mode interrupts vs. Real mode interrupts.	828
54.10	Handling interrupts with DPML.	828
54.11	Interrupt redirection.	828
54.12	Processor access.	828
54.13	I/O port access.	828
54.14	dos memory access.	828
54.15	FPC specialities.	829
54.16	Selectors and descriptors.	829
54.17	What is DPML.	829
54.18	Constants, types and variables	829
54.18.1	Constants	829
54.18.2	Types	832
54.18.3	Variables	834
54.19	Procedures and functions	834
54.19.1	allocate_ldt_descriptors	834

54.19.2	allocate_memory_block	837
54.19.3	copyfromdos	838
54.19.4	copytodos	838
54.19.5	create_code_segment_alias_descriptor	839
54.19.6	disable	839
54.19.7	dpmi_dosmemfillchar	839
54.19.8	dpmi_dosmemfillword	840
54.19.9	dpmi_dosmemget	840
54.19.10	dpmi_dosmemmove	840
54.19.11	dpmi_dosmemput	840
54.19.12	enable	841
54.19.13	free_ldt_descriptor	841
54.19.14	free_linear_addr_mapping	841
54.19.15	free_memory_block	841
54.19.16	free_rm_callback	842
54.19.17	get_cs	842
54.19.18	get_descriptor_access_right	842
54.19.19	get_dpmi_version	843
54.19.20	get_ds	843
54.19.21	get_exception_handler	843
54.19.22	get_linear_addr	844
54.19.23	get_meminfo	844
54.19.24	get_next_selector_increment_value	845
54.19.25	get_page_attributes	846
54.19.26	get_page_size	846
54.19.27	get_pm_exception_handler	846
54.19.28	get_pm_interrupt	846
54.19.29	get_rm_callback	847
54.19.30	get_rm_interrupt	851
54.19.31	get_run_mode	852
54.19.32	get_segment_base_address	853
54.19.33	get_segment_limit	853
54.19.34	get_ss	853
54.19.35	global_dos_alloc	854
54.19.36	global_dos_free	856
54.19.37	inportb	857
54.19.38	inportl	857
54.19.39	inportw	857
54.19.40	lock_code	858
54.19.41	lock_data	858

54.19.42	lock_linear_region	858
54.19.43	map_device_in_memory_block	859
54.19.44	outportb	859
54.19.45	outportl	860
54.19.46	outportw	860
54.19.47	realintr	860
54.19.48	request_linear_region	861
54.19.49	segment_to_descriptor	862
54.19.50	seg_fillchar	862
54.19.51	seg_fillword	863
54.19.52	seg_move	864
54.19.53	set_descriptor_access_right	864
54.19.54	set_exception_handler	864
54.19.55	set_page_attributes	865
54.19.56	set_pm_exception_handler	865
54.19.57	set_pm_interrupt	865
54.19.58	set_rm_interrupt	867
54.19.59	set_segment_base_address	867
54.19.60	set_segment_limit	867
54.19.61	tb_offset	868
54.19.62	tb_segment	868
54.19.63	tb_size	868
54.19.64	transfer_buffer	869
54.19.65	unlock_code	869
54.19.66	unlock_data	869
54.19.67	unlock_linear_region	870
54.20	tdpmiversioninfo	870
54.21	tmeminfo	871
54.22	tseginfo	871
55	Reference for unit 'gpm'	872
55.1	Used units	872
55.2	Overview	872
55.3	Constants, types and variables	872
55.3.1	Constants	872
55.3.2	Types	874
55.3.3	Variables	875
55.4	Procedures and functions	876
55.4.1	Gpm_AnyDouble	876
55.4.2	Gpm_AnySingle	876

55.4.3	Gpm_AnyTriple	876
55.4.4	gpm_close	877
55.4.5	gpm_fitvalues	877
55.4.6	gpm_fitvaluesM	877
55.4.7	gpm_getevent	878
55.4.8	gpm_getsnapshot	879
55.4.9	gpm_lowerroi	879
55.4.10	gpm_open	879
55.4.11	gpm_poproi	880
55.4.12	gpm_pushroi	880
55.4.13	gpm_raiseroi	880
55.4.14	gpm_repeat	881
55.4.15	Gpm_StrictDouble	881
55.4.16	Gpm_StrictSingle	881
55.4.17	Gpm_StrictTriple	881
55.5	Tgpm_connect	882
55.6	Tgpm_event	882
55.7	Tgpm_roi	882
56	Reference for unit 'Graph'	884
56.1	Used units	884
56.2	Overview	884
56.3	Categorized functions: Text and font handling.	884
56.4	Categorized functions: Filled drawings.	885
56.5	Categorized functions: Drawing primitives.	885
56.6	Categorized functions: Color management.	886
56.7	Categorized functions: Screen management.	886
56.8	Categorized functions: Initialization.	887
56.9	Target specific issues: Linux.	887
56.10	Target specific issues: DOS.	888
56.11	A word about mode selection.	888
56.12	Requirements.	893
56.13	Constants, types and variables	893
56.13.1	Constants	893
56.13.2	Types	910
56.13.3	Variables	913
56.14	Procedures and functions	916
56.14.1	Arc	916
56.14.2	Bar	916
56.14.3	Bar3D	916

56.14.4	ClearDevice	917
56.14.5	Closegraph	917
56.14.6	DetectGraph	917
56.14.7	DrawPoly	917
56.14.8	Ellipse	918
56.14.9	FillEllipse	918
56.14.10	FillPoly	918
56.14.11	FloodFill	918
56.14.12	GetArcCoords	919
56.14.13	GetAspectRatio	919
56.14.14	GetColor	919
56.14.15	GetDefaultPalette	919
56.14.16	GetDirectVideo	920
56.14.17	GetDriverName	920
56.14.18	GetFillPattern	920
56.14.19	GetFillSettings	920
56.14.20	GetGraphMode	921
56.14.21	GetLineSettings	921
56.14.22	GetMaxColor	921
56.14.23	GetMaxMode	921
56.14.24	GetMaxX	922
56.14.25	GetMaxY	922
56.14.26	GetModeName	922
56.14.27	GetModeRange	922
56.14.28	GetPalette	923
56.14.29	GetPaletteSize	923
56.14.30	GetTextSettings	923
56.14.31	GetViewSettings	923
56.14.32	GetX	924
56.14.33	GetY	924
56.14.34	GraphDefaults	924
56.14.35	GraphErrorMsg	924
56.14.36	GraphResult	925
56.14.37	InitGraph	925
56.14.38	InstallUserDriver	926
56.14.39	InstallUserFont	926
56.14.40	LineRel	926
56.14.41	LineTo	927
56.14.42	MoveRel	927
56.14.43	MoveTo	927

56.14.44 OutText	927
56.14.45 PieSlice	928
56.14.46 queryadapterinfo	928
56.14.47 Rectangle	928
56.14.48 RegisterBGIDriver	928
56.14.49 RegisterBGIfont	929
56.14.50 RestoreCrtMode	929
56.14.51 Sector	929
56.14.52 SetAspectRatio	929
56.14.53 SetColor	930
56.14.54 SetDirectVideo	930
56.14.55 SetFillPattern	930
56.14.56 SetFillStyle	930
56.14.57 SetGraphMode	931
56.14.58 SetLineStyle	931
56.14.59 SetPalette	932
56.14.60 SetTextJustify	932
56.14.61 SetTextStyle	932
56.14.62 SetUserCharSize	933
56.14.63 SetViewPort	933
56.14.64 SetWriteMode	934
56.14.65 SetWriteModeEx	934
56.14.66 TextHeight	934
56.14.67 TextWidth	934
56.15 ArcCoordsType	935
56.16 FillSettingsType	935
56.17 LineSettingsType	935
56.18 PaletteType	935
56.19 PointType	935
56.20 RGBRec	936
56.21 TextSettingsType	936
56.22 TModeInfo	936
56.23 TResolutionRec	937
56.24 ViewPortType	937
57 Reference for unit 'heaptrc'	939
57.1 Used units	939
57.2 Overview	939
57.3 Controlling HeapTrc with environment variables.	940
57.4 HeapTrc Usage.	940

57.5	Constants, types and variables	941
57.5.1	Constants	941
57.5.2	Types	943
57.6	Procedures and functions	943
57.6.1	CheckPointer	943
57.6.2	DumpHeap	943
57.6.3	SetHeapExtraInfo	944
57.6.4	SetHeapTraceOutput	945
58	Reference for unit 'ipc'	946
58.1	Used units	946
58.2	Overview	946
58.3	Constants, types and variables	946
58.3.1	Constants	946
58.3.2	Types	949
58.4	Procedures and functions	950
58.4.1	ftok	950
58.4.2	msgctl	951
58.4.3	msgget	953
58.4.4	msgrcv	954
58.4.5	msgsnd	954
58.4.6	semctl	955
58.4.7	semget	960
58.4.8	semop	960
58.4.9	semtimedop	961
58.4.10	shmat	961
58.4.11	shmctl	962
58.4.12	shmdt	964
58.4.13	shmget	964
58.5	TIPC_Perm	965
58.6	TMSG	965
58.7	TMSGbuf	965
58.8	TMSGinfo	965
58.9	TMSQid_ds	966
58.10	TSEMbuf	966
58.11	TSEMid_ds	966
58.12	TSEMinfo	967
58.13	TShmid_ds	967
58.14	TSHMinfo	968
58.15	TSHM_info	968

59	Reference for unit 'keyboard'	969
59.1	Used units	969
59.2	Overview	969
59.3	Unix specific notes.	969
59.4	Writing a keyboard driver.	971
59.5	Keyboard scan codes.	974
59.6	Constants, types and variables	976
59.6.1	Constants	976
59.6.2	Types	981
59.7	Procedures and functions	981
59.7.1	DoneKeyboard	981
59.7.2	FunctionKeyName	982
59.7.3	GetKeyboardDriver	982
59.7.4	GetKeyEvent	983
59.7.5	GetKeyEventChar	983
59.7.6	GetKeyEventCode	984
59.7.7	GetKeyEventFlags	985
59.7.8	GetKeyEventShiftState	985
59.7.9	GetKeyEventUniCode	986
59.7.10	InitKeyboard	986
59.7.11	IsFunctionKey	986
59.7.12	KeyEventToString	987
59.7.13	KeyPressed	987
59.7.14	PollKeyEvent	988
59.7.15	PollShiftStateEvent	988
59.7.16	PutKeyEvent	989
59.7.17	SetKeyboardDriver	990
59.7.18	ShiftStateToString	990
59.7.19	TranslateKeyEvent	991
59.7.20	TranslateKeyEventUniCode	991
59.8	TKeyboardDriver	991
59.9	TKeyRecord	992
59.10	TTreeElement	992
60	Reference for unit 'lineinfo'	994
60.1	Used units	994
60.2	Overview	994
60.3	Constants, types and variables	994
60.3.1	Types	994
60.3.2	Variables	994

60.4	Procedures and functions	995
60.4.1	CloseStabs	995
60.4.2	GetLineInfo	995
60.4.3	StabBackTraceStr	995
61	Reference for unit 'Linux'	996
61.1	Used units	996
61.2	Overview	996
61.3	Constants, types and variables	996
61.3.1	Constants	996
61.3.2	Types	1012
61.4	Procedures and functions	1014
61.4.1	capget	1014
61.4.2	capset	1014
61.4.3	clock_getres	1015
61.4.4	clock_gettime	1015
61.4.5	clock_settime	1015
61.4.6	clone	1015
61.4.7	epoll_create	1016
61.4.8	epoll_ctl	1017
61.4.9	epoll_wait	1017
61.4.10	fdatasync	1018
61.4.11	futex	1018
61.4.12	futex_op	1019
61.4.13	futimens	1019
61.4.14	inotify_add_watch	1019
61.4.15	inotify_init	1020
61.4.16	inotify_init1	1020
61.4.17	inotify_rm_watch	1020
61.4.18	modify_ldt	1021
61.4.19	sched_yield	1021
61.4.20	setregid	1021
61.4.21	setreuid	1021
61.4.22	statx	1021
61.4.23	sync_file_range	1022
61.4.24	Sysinfo	1022
61.4.25	utimensat	1023
61.5	EPoll_Event	1023
61.6	inotify_event	1024
61.7	kernel_timespec	1024

61.8	statx_timestamp	1024
61.9	tstatx	1024
61.10	TSysInfo	1025
61.11	user_cap_data	1025
61.12	user_cap_header	1026
61.13	user_desc	1026
62	Reference for unit 'Infodwrf'	1027
62.1	Used units	1027
62.2	Overview	1027
62.3	Constants, types and variables	1027
62.3.1	Types	1027
62.3.2	Variables	1027
62.4	Procedures and functions	1028
62.4.1	CloseDwarf	1028
62.4.2	DwarfBackTraceStr	1028
62.4.3	GetLineInfo	1028
63	Reference for unit 'Math'	1029
63.1	Used units	1029
63.2	Overview	1029
63.3	Cash flow functions.	1029
63.4	Geometrical functions.	1030
63.5	Statistical functions.	1030
63.6	Number converting.	1031
63.7	Exponential and logarithmic functions.	1031
63.8	Hyperbolic functions.	1031
63.9	Trigonometric functions.	1032
63.10	Angle unit conversion.	1032
63.11	Min/max determination.	1032
63.12	Constants, types and variables	1033
63.12.1	Constants	1033
63.12.2	Types	1034
63.13	Procedures and functions	1035
63.13.1	ArcCos	1035
63.13.2	ArcCosH	1035
63.13.3	ArcCot	1036
63.13.4	ArcCotH	1036
63.13.5	ArcCsc	1036
63.13.6	ArcCscH	1036
63.13.7	ArCosH	1036

63.13.8	ArcSec	1037
63.13.9	ArcSecH	1037
63.13.10	ArcSin	1037
63.13.11	ArcSinH	1038
63.13.12	ArcTan2	1038
63.13.13	ArcTanH	1038
63.13.14	ArSinH	1039
63.13.15	ArTanH	1039
63.13.16	Ceil	1040
63.13.17	Ceil64	1040
63.13.18	ClearExceptions	1040
63.13.19	CompareValue	1041
63.13.20	Cosecant	1041
63.13.21	cosh	1041
63.13.22	Cot	1042
63.13.23	Cotan	1042
63.13.24	CotH	1043
63.13.25	Csc	1043
63.13.26	CscH	1043
63.13.27	CycleToDeg	1043
63.13.28	CycleToGrad	1043
63.13.29	CycleToRad	1043
63.13.30	DegNormalize	1044
63.13.31	DegToCycle	1044
63.13.32	DegToGrad	1044
63.13.33	DegToRad	1045
63.13.34	DivMod	1045
63.13.35	EnsureRange	1046
63.13.36	ExpM1	1046
63.13.37	Floor	1046
63.13.38	Floor64	1047
63.13.39	FMod	1047
63.13.40	Frexp	1047
63.13.41	FutureValue	1048
63.13.42	GetExceptionMask	1048
63.13.43	GetPrecisionMode	1048
63.13.44	GetRoundMode	1049
63.13.45	GradToCycle	1049
63.13.46	GradToDeg	1049
63.13.47	GradToRad	1049

63.13.48 Hypot	1050
63.13.49 IfThen	1050
63.13.50 InRange	1051
63.13.51 InterestRate	1051
63.13.52 IntPower	1051
63.13.53 IsInfinite	1052
63.13.54 IsNan	1052
63.13.55 IsZero	1052
63.13.56 Ldexp	1053
63.13.57 LnXP1	1053
63.13.58 Log10	1054
63.13.59 Log2	1054
63.13.60 LogN	1055
63.13.61 Max	1056
63.13.62 MaxIntValue	1056
63.13.63 MaxValue	1057
63.13.64 Mean	1058
63.13.65 MeanAndStdDev	1059
63.13.66 Min	1060
63.13.67 MinIntValue	1060
63.13.68 MinValue	1061
63.13.69 modulus(Float,Float):Float	1062
63.13.70 MomentSkewKurtosis	1062
63.13.71 Norm	1064
63.13.72 NumberOfPeriods	1064
63.13.73 Payment	1065
63.13.74 PopnStdDev	1065
63.13.75 PopnVariance	1066
63.13.76 Power	1067
63.13.77 power(Float,Float):Float	1067
63.13.78 power(Int64,Int64):Int64	1067
63.13.79 PresentValue	1068
63.13.80 RadToCycle	1068
63.13.81 RadToDeg	1069
63.13.82 RadToGrad	1069
63.13.83 RandG	1070
63.13.84 RandomFrom	1070
63.13.85 RandomRange	1070
63.13.86 RoundTo	1071
63.13.87 SameValue	1071

63.13.88	Sec	1072
63.13.89	Secant	1072
63.13.90	SecH	1072
63.13.91	SetExceptionMask	1072
63.13.92	SetPrecisionMode	1072
63.13.93	SetRoundMode	1073
63.13.94	Sign	1073
63.13.95	SimpleRoundTo	1073
63.13.96	SinCos	1074
63.13.97	sinh	1074
63.13.98	StdDev	1075
63.13.99	Sum	1076
63.13.100	SumInt	1076
63.13.101	SumOfSquares	1077
63.13.102	SumsAndSquares	1077
63.13.103	Tan	1078
63.13.104	tanh	1079
63.13.105	TotalVariance	1080
63.13.106	Variance	1080
63.14	EInvalidArgument	1082
63.14.1	Description	1082
64	Reference for unit 'matrix'	1083
64.1	Used units	1083
64.2	Overview	1083
64.3	Constants, types and variables	1084
64.3.1	Types	1084
64.4	Procedures and functions	1086
64.4.1	add(Tmatrix2_double,Double):Tmatrix2_double	1086
64.4.2	add(Tmatrix2_double,Tmatrix2_double):Tmatrix2_double	1086
64.4.3	add(Tmatrix2_extended,extended):Tmatrix2_extended	1087
64.4.4	add(Tmatrix2_extended,Tmatrix2_extended):Tmatrix2_extended	1087
64.4.5	add(Tmatrix2_single,single):Tmatrix2_single	1087
64.4.6	add(Tmatrix2_single,Tmatrix2_single):Tmatrix2_single	1087
64.4.7	add(Tmatrix3_double,Double):Tmatrix3_double	1087
64.4.8	add(Tmatrix3_double,Tmatrix3_double):Tmatrix3_double	1088
64.4.9	add(Tmatrix3_extended,extended):Tmatrix3_extended	1088
64.4.10	add(Tmatrix3_extended,Tmatrix3_extended):Tmatrix3_extended	1088
64.4.11	add(Tmatrix3_single,single):Tmatrix3_single	1088
64.4.12	add(Tmatrix3_single,Tmatrix3_single):Tmatrix3_single	1088

64.4.13	add(Tmatrix4_double,Double):Tmatrix4_double	1089
64.4.14	add(Tmatrix4_double,Tmatrix4_double):Tmatrix4_double	1089
64.4.15	add(Tmatrix4_extended,extended):Tmatrix4_extended	1089
64.4.16	add(Tmatrix4_extended,Tmatrix4_extended):Tmatrix4_extended	1089
64.4.17	add(Tmatrix4_single,single):Tmatrix4_single	1089
64.4.18	add(Tmatrix4_single,Tmatrix4_single):Tmatrix4_single	1090
64.4.19	add(Tvector2_double,Double):Tvector2_double	1090
64.4.20	add(Tvector2_double,Tvector2_double):Tvector2_double	1090
64.4.21	add(Tvector2_extended,extended):Tvector2_extended	1090
64.4.22	add(Tvector2_extended,Tvector2_extended):Tvector2_extended	1090
64.4.23	add(Tvector2_single,single):Tvector2_single	1091
64.4.24	add(Tvector2_single,Tvector2_single):Tvector2_single	1091
64.4.25	add(Tvector3_double,Double):Tvector3_double	1091
64.4.26	add(Tvector3_double,Tvector3_double):Tvector3_double	1091
64.4.27	add(Tvector3_extended,extended):Tvector3_extended	1091
64.4.28	add(Tvector3_extended,Tvector3_extended):Tvector3_extended	1092
64.4.29	add(Tvector3_single,single):Tvector3_single	1092
64.4.30	add(Tvector3_single,Tvector3_single):Tvector3_single	1092
64.4.31	add(Tvector4_double,Double):Tvector4_double	1092
64.4.32	add(Tvector4_double,Tvector4_double):Tvector4_double	1092
64.4.33	add(Tvector4_extended,extended):Tvector4_extended	1093
64.4.34	add(Tvector4_extended,Tvector4_extended):Tvector4_extended	1093
64.4.35	add(Tvector4_single,single):Tvector4_single	1093
64.4.36	add(Tvector4_single,Tvector4_single):Tvector4_single	1093
64.4.37	assign(Tmatrix2_double):Tmatrix2_extended	1093
64.4.38	assign(Tmatrix2_double):Tmatrix2_single	1094
64.4.39	assign(Tmatrix2_double):Tmatrix3_double	1094
64.4.40	assign(Tmatrix2_double):Tmatrix3_extended	1094
64.4.41	assign(Tmatrix2_double):Tmatrix3_single	1094
64.4.42	assign(Tmatrix2_double):Tmatrix4_double	1094
64.4.43	assign(Tmatrix2_double):Tmatrix4_extended	1095
64.4.44	assign(Tmatrix2_double):Tmatrix4_single	1095
64.4.45	assign(Tmatrix2_extended):Tmatrix2_double	1095
64.4.46	assign(Tmatrix2_extended):Tmatrix2_single	1095
64.4.47	assign(Tmatrix2_extended):Tmatrix3_double	1096
64.4.48	assign(Tmatrix2_extended):Tmatrix3_extended	1096
64.4.49	assign(Tmatrix2_extended):Tmatrix3_single	1096
64.4.50	assign(Tmatrix2_extended):Tmatrix4_double	1096
64.4.51	assign(Tmatrix2_extended):Tmatrix4_extended	1097
64.4.52	assign(Tmatrix2_extended):Tmatrix4_single	1097

64.4.53	assign(Tmatrix2_single):Tmatrix2_double	1097
64.4.54	assign(Tmatrix2_single):Tmatrix2_extended	1097
64.4.55	assign(Tmatrix2_single):Tmatrix3_double	1097
64.4.56	assign(Tmatrix2_single):Tmatrix3_extended	1098
64.4.57	assign(Tmatrix2_single):Tmatrix3_single	1098
64.4.58	assign(Tmatrix2_single):Tmatrix4_double	1098
64.4.59	assign(Tmatrix2_single):Tmatrix4_extended	1098
64.4.60	assign(Tmatrix2_single):Tmatrix4_single	1098
64.4.61	assign(Tmatrix3_double):Tmatrix2_double	1099
64.4.62	assign(Tmatrix3_double):Tmatrix2_extended	1099
64.4.63	assign(Tmatrix3_double):Tmatrix2_single	1099
64.4.64	assign(Tmatrix3_double):Tmatrix3_extended	1099
64.4.65	assign(Tmatrix3_double):Tmatrix3_single	1099
64.4.66	assign(Tmatrix3_double):Tmatrix4_double	1100
64.4.67	assign(Tmatrix3_double):Tmatrix4_extended	1100
64.4.68	assign(Tmatrix3_double):Tmatrix4_single	1100
64.4.69	assign(Tmatrix3_extended):Tmatrix2_double	1100
64.4.70	assign(Tmatrix3_extended):Tmatrix2_extended	1101
64.4.71	assign(Tmatrix3_extended):Tmatrix2_single	1101
64.4.72	assign(Tmatrix3_extended):Tmatrix3_double	1101
64.4.73	assign(Tmatrix3_extended):Tmatrix3_single	1101
64.4.74	assign(Tmatrix3_extended):Tmatrix4_double	1102
64.4.75	assign(Tmatrix3_extended):Tmatrix4_extended	1102
64.4.76	assign(Tmatrix3_extended):Tmatrix4_single	1102
64.4.77	assign(Tmatrix3_single):Tmatrix2_double	1102
64.4.78	assign(Tmatrix3_single):Tmatrix2_extended	1102
64.4.79	assign(Tmatrix3_single):Tmatrix2_single	1103
64.4.80	assign(Tmatrix3_single):Tmatrix3_double	1103
64.4.81	assign(Tmatrix3_single):Tmatrix3_extended	1103
64.4.82	assign(Tmatrix3_single):Tmatrix4_double	1103
64.4.83	assign(Tmatrix3_single):Tmatrix4_extended	1104
64.4.84	assign(Tmatrix3_single):Tmatrix4_single	1104
64.4.85	assign(Tmatrix4_double):Tmatrix2_double	1104
64.4.86	assign(Tmatrix4_double):Tmatrix2_extended	1104
64.4.87	assign(Tmatrix4_double):Tmatrix2_single	1104
64.4.88	assign(Tmatrix4_double):Tmatrix3_double	1105
64.4.89	assign(Tmatrix4_double):Tmatrix3_extended	1105
64.4.90	assign(Tmatrix4_double):Tmatrix3_single	1105
64.4.91	assign(Tmatrix4_double):Tmatrix4_extended	1105
64.4.92	assign(Tmatrix4_double):Tmatrix4_single	1105

64.4.93	assign(Tmatrix4_extended):Tmatrix2_double	1106
64.4.94	assign(Tmatrix4_extended):Tmatrix2_extended	1106
64.4.95	assign(Tmatrix4_extended):Tmatrix2_single	1106
64.4.96	assign(Tmatrix4_extended):Tmatrix3_double	1106
64.4.97	assign(Tmatrix4_extended):Tmatrix3_extended	1107
64.4.98	assign(Tmatrix4_extended):Tmatrix3_single	1107
64.4.99	assign(Tmatrix4_extended):Tmatrix4_double	1107
64.4.100	assign(Tmatrix4_extended):Tmatrix4_single	1107
64.4.101	assign(Tmatrix4_single):Tmatrix2_double	1107
64.4.102	assign(Tmatrix4_single):Tmatrix2_extended	1108
64.4.103	assign(Tmatrix4_single):Tmatrix2_single	1108
64.4.104	assign(Tmatrix4_single):Tmatrix3_double	1108
64.4.105	assign(Tmatrix4_single):Tmatrix3_extended	1108
64.4.106	assign(Tmatrix4_single):Tmatrix3_single	1109
64.4.107	assign(Tmatrix4_single):Tmatrix4_double	1109
64.4.108	assign(Tmatrix4_single):Tmatrix4_extended	1109
64.4.109	assign(Tvector2_double):Tvector2_extended	1109
64.4.110	assign(Tvector2_double):Tvector2_single	1109
64.4.111	assign(Tvector2_double):Tvector3_double	1110
64.4.112	assign(Tvector2_double):Tvector3_extended	1110
64.4.113	assign(Tvector2_double):Tvector3_single	1110
64.4.114	assign(Tvector2_double):Tvector4_double	1110
64.4.115	assign(Tvector2_double):Tvector4_extended	1110
64.4.116	assign(Tvector2_double):Tvector4_single	1111
64.4.117	assign(Tvector2_extended):Tvector2_double	1111
64.4.118	assign(Tvector2_extended):Tvector2_single	1111
64.4.119	assign(Tvector2_extended):Tvector3_double	1111
64.4.120	assign(Tvector2_extended):Tvector3_extended	1112
64.4.121	assign(Tvector2_extended):Tvector3_single	1112
64.4.122	assign(Tvector2_extended):Tvector4_double	1112
64.4.123	assign(Tvector2_extended):Tvector4_extended	1112
64.4.124	assign(Tvector2_extended):Tvector4_single	1113
64.4.125	assign(Tvector2_single):Tvector2_double	1113
64.4.126	assign(Tvector2_single):Tvector2_extended	1113
64.4.127	assign(Tvector2_single):Tvector3_double	1113
64.4.128	assign(Tvector2_single):Tvector3_extended	1113
64.4.129	assign(Tvector2_single):Tvector3_single	1114
64.4.130	assign(Tvector2_single):Tvector4_double	1114
64.4.131	assign(Tvector2_single):Tvector4_extended	1114
64.4.132	assign(Tvector2_single):Tvector4_single	1114

64.4.133	assign(Tvector3_double):Tvector2_double	1115
64.4.134	assign(Tvector3_double):Tvector2_extended	1115
64.4.135	assign(Tvector3_double):Tvector2_single	1115
64.4.136	assign(Tvector3_double):Tvector3_extended	1115
64.4.137	assign(Tvector3_double):Tvector3_single	1115
64.4.138	assign(Tvector3_double):Tvector4_double	1116
64.4.139	assign(Tvector3_double):Tvector4_extended	1116
64.4.140	assign(Tvector3_double):Tvector4_single	1116
64.4.141	assign(Tvector3_extended):Tvector2_double	1116
64.4.142	assign(Tvector3_extended):Tvector2_extended	1117
64.4.143	assign(Tvector3_extended):Tvector2_single	1117
64.4.144	assign(Tvector3_extended):Tvector3_double	1117
64.4.145	assign(Tvector3_extended):Tvector3_single	1117
64.4.146	assign(Tvector3_extended):Tvector4_double	1118
64.4.147	assign(Tvector3_extended):Tvector4_extended	1118
64.4.148	assign(Tvector3_extended):Tvector4_single	1118
64.4.149	assign(Tvector3_single):Tvector2_double	1118
64.4.150	assign(Tvector3_single):Tvector2_extended	1118
64.4.151	assign(Tvector3_single):Tvector2_single	1119
64.4.152	assign(Tvector3_single):Tvector3_double	1119
64.4.153	assign(Tvector3_single):Tvector3_extended	1119
64.4.154	assign(Tvector3_single):Tvector4_double	1119
64.4.155	assign(Tvector3_single):Tvector4_extended	1120
64.4.156	assign(Tvector3_single):Tvector4_single	1120
64.4.157	assign(Tvector4_double):Tvector2_double	1120
64.4.158	assign(Tvector4_double):Tvector2_extended	1120
64.4.159	assign(Tvector4_double):Tvector2_single	1120
64.4.160	assign(Tvector4_double):Tvector3_double	1121
64.4.161	assign(Tvector4_double):Tvector3_extended	1121
64.4.162	assign(Tvector4_double):Tvector3_single	1121
64.4.163	assign(Tvector4_double):Tvector4_extended	1121
64.4.164	assign(Tvector4_double):Tvector4_single	1122
64.4.165	assign(Tvector4_extended):Tvector2_double	1122
64.4.166	assign(Tvector4_extended):Tvector2_extended	1122
64.4.167	assign(Tvector4_extended):Tvector2_single	1122
64.4.168	assign(Tvector4_extended):Tvector3_double	1123
64.4.169	assign(Tvector4_extended):Tvector3_extended	1123
64.4.170	assign(Tvector4_extended):Tvector3_single	1123
64.4.171	assign(Tvector4_extended):Tvector4_double	1123
64.4.172	assign(Tvector4_extended):Tvector4_single	1124

64.4.173	assign(Tvector4_single):Tvector2_double	1124
64.4.174	assign(Tvector4_single):Tvector2_extended	1124
64.4.175	assign(Tvector4_single):Tvector2_single	1124
64.4.176	assign(Tvector4_single):Tvector3_double	1125
64.4.177	assign(Tvector4_single):Tvector3_extended	1125
64.4.178	assign(Tvector4_single):Tvector3_single	1125
64.4.179	assign(Tvector4_single):Tvector4_double	1125
64.4.180	assign(Tvector4_single):Tvector4_extended	1125
64.4.181	divide(Tmatrix2_double,Double):Tmatrix2_double	1126
64.4.182	divide(Tmatrix2_extended,extended):Tmatrix2_extended	1126
64.4.183	divide(Tmatrix2_single,single):Tmatrix2_single	1126
64.4.184	divide(Tmatrix3_double,Double):Tmatrix3_double	1126
64.4.185	divide(Tmatrix3_extended,extended):Tmatrix3_extended	1126
64.4.186	divide(Tmatrix3_single,single):Tmatrix3_single	1127
64.4.187	divide(Tmatrix4_double,Double):Tmatrix4_double	1127
64.4.188	divide(Tmatrix4_extended,extended):Tmatrix4_extended	1127
64.4.189	divide(Tmatrix4_single,single):Tmatrix4_single	1127
64.4.190	divide(Tvector2_double,Double):Tvector2_double	1127
64.4.191	divide(Tvector2_extended,extended):Tvector2_extended	1128
64.4.192	divide(Tvector2_single,single):Tvector2_single	1128
64.4.193	divide(Tvector3_double,Double):Tvector3_double	1128
64.4.194	divide(Tvector3_extended,extended):Tvector3_extended	1128
64.4.195	divide(Tvector3_single,single):Tvector3_single	1128
64.4.196	divide(Tvector4_double,Double):Tvector4_double	1129
64.4.197	divide(Tvector4_extended,extended):Tvector4_extended	1129
64.4.198	divide(Tvector4_single,single):Tvector4_single	1129
64.4.199	multiply(Tmatrix2_double,Double):Tmatrix2_double	1129
64.4.200	multiply(Tmatrix2_double,Tmatrix2_double):Tmatrix2_double	1129
64.4.201	multiply(Tmatrix2_double,Tvector2_double):Tvector2_double	1130
64.4.202	multiply(Tmatrix2_extended,extended):Tmatrix2_extended	1130
64.4.203	multiply(Tmatrix2_extended,Tmatrix2_extended):Tmatrix2_extended	1130
64.4.204	multiply(Tmatrix2_extended,Tvector2_extended):Tvector2_extended	1130
64.4.205	multiply(Tmatrix2_single,single):Tmatrix2_single	1131
64.4.206	multiply(Tmatrix2_single,Tmatrix2_single):Tmatrix2_single	1131
64.4.207	multiply(Tmatrix2_single,Tvector2_single):Tvector2_single	1131
64.4.208	multiply(Tmatrix3_double,Double):Tmatrix3_double	1131
64.4.209	multiply(Tmatrix3_double,Tmatrix3_double):Tmatrix3_double	1131
64.4.210	multiply(Tmatrix3_double,Tvector3_double):Tvector3_double	1132
64.4.211	multiply(Tmatrix3_extended,extended):Tmatrix3_extended	1132
64.4.212	multiply(Tmatrix3_extended,Tmatrix3_extended):Tmatrix3_extended	1132

64.4.213	multiply(Tmatrix3_extended,Tvector3_extended):Tvector3_extended . . .	1132
64.4.214	multiply(Tmatrix3_single,single):Tmatrix3_single	1133
64.4.215	multiply(Tmatrix3_single,Tmatrix3_single):Tmatrix3_single	1133
64.4.216	multiply(Tmatrix3_single,Tvector3_single):Tvector3_single	1133
64.4.217	multiply(Tmatrix4_double,Double):Tmatrix4_double	1133
64.4.218	multiply(Tmatrix4_double,Tmatrix4_double):Tmatrix4_double	1133
64.4.219	multiply(Tmatrix4_double,Tvector4_double):Tvector4_double	1134
64.4.220	multiply(Tmatrix4_extended,extended):Tmatrix4_extended	1134
64.4.221	multiply(Tmatrix4_extended,Tmatrix4_extended):Tmatrix4_extended . .	1134
64.4.222	multiply(Tmatrix4_extended,Tvector4_extended):Tvector4_extended . . .	1134
64.4.223	multiply(Tmatrix4_single,single):Tmatrix4_single	1135
64.4.224	multiply(Tmatrix4_single,Tmatrix4_single):Tmatrix4_single	1135
64.4.225	multiply(Tmatrix4_single,Tvector4_single):Tvector4_single	1135
64.4.226	multiply(Tvector2_double,Double):Tvector2_double	1135
64.4.227	multiply(Tvector2_double,Tvector2_double):Tvector2_double	1135
64.4.228	multiply(Tvector2_extended,extended):Tvector2_extended	1136
64.4.229	multiply(Tvector2_extended,Tvector2_extended):Tvector2_extended . . .	1136
64.4.230	multiply(Tvector2_single,single):Tvector2_single	1136
64.4.231	multiply(Tvector2_single,Tvector2_single):Tvector2_single	1136
64.4.232	multiply(Tvector3_double,Double):Tvector3_double	1136
64.4.233	multiply(Tvector3_double,Tvector3_double):Tvector3_double	1137
64.4.234	multiply(Tvector3_extended,extended):Tvector3_extended	1137
64.4.235	multiply(Tvector3_extended,Tvector3_extended):Tvector3_extended . . .	1137
64.4.236	multiply(Tvector3_single,single):Tvector3_single	1137
64.4.237	multiply(Tvector3_single,Tvector3_single):Tvector3_single	1137
64.4.238	multiply(Tvector4_double,Double):Tvector4_double	1138
64.4.239	multiply(Tvector4_double,Tvector4_double):Tvector4_double	1138
64.4.240	multiply(Tvector4_extended,extended):Tvector4_extended	1138
64.4.241	multiply(Tvector4_extended,Tvector4_extended):Tvector4_extended . . .	1138
64.4.242	multiply(Tvector4_single,single):Tvector4_single	1138
64.4.243	multiply(Tvector4_single,Tvector4_single):Tvector4_single	1139
64.4.244	negative(Tmatrix2_double):Tmatrix2_double	1139
64.4.245	negative(Tmatrix2_extended):Tmatrix2_extended	1139
64.4.246	negative(Tmatrix2_single):Tmatrix2_single	1139
64.4.247	negative(Tmatrix3_double):Tmatrix3_double	1139
64.4.248	negative(Tmatrix3_extended):Tmatrix3_extended	1139
64.4.249	negative(Tmatrix3_single):Tmatrix3_single	1140
64.4.250	negative(Tmatrix4_double):Tmatrix4_double	1140
64.4.251	negative(Tmatrix4_extended):Tmatrix4_extended	1140
64.4.252	negative(Tmatrix4_single):Tmatrix4_single	1140

64.4.253	negative(Tvector2_double):Tvector2_double	1140
64.4.254	negative(Tvector2_extended):Tvector2_extended	1140
64.4.255	negative(Tvector2_single):Tvector2_single	1141
64.4.256	negative(Tvector3_double):Tvector3_double	1141
64.4.257	negative(Tvector3_extended):Tvector3_extended	1141
64.4.258	negative(Tvector3_single):Tvector3_single	1141
64.4.259	negative(Tvector4_double):Tvector4_double	1141
64.4.260	negative(Tvector4_extended):Tvector4_extended	1142
64.4.261	negative(Tvector4_single):Tvector4_single	1142
64.4.262	power(Tvector2_double,Tvector2_double):Double	1142
64.4.263	power(Tvector2_extended,Tvector2_extended):extended	1142
64.4.264	power(Tvector2_single,Tvector2_single):single	1142
64.4.265	power(Tvector3_double,Tvector3_double):Double	1143
64.4.266	power(Tvector3_extended,Tvector3_extended):extended	1143
64.4.267	power(Tvector3_single,Tvector3_single):single	1143
64.4.268	power(Tvector4_double,Tvector4_double):Double	1143
64.4.269	power(Tvector4_extended,Tvector4_extended):extended	1143
64.4.270	power(Tvector4_single,Tvector4_single):single	1144
64.4.271	subtract(Tmatrix2_double,Double):Tmatrix2_double	1144
64.4.272	subtract(Tmatrix2_double,Tmatrix2_double):Tmatrix2_double	1144
64.4.273	subtract(Tmatrix2_extended,extended):Tmatrix2_extended	1144
64.4.274	subtract(Tmatrix2_extended,Tmatrix2_extended):Tmatrix2_extended	1144
64.4.275	subtract(Tmatrix2_single,single):Tmatrix2_single	1145
64.4.276	subtract(Tmatrix2_single,Tmatrix2_single):Tmatrix2_single	1145
64.4.277	subtract(Tmatrix3_double,Double):Tmatrix3_double	1145
64.4.278	subtract(Tmatrix3_double,Tmatrix3_double):Tmatrix3_double	1145
64.4.279	subtract(Tmatrix3_extended,extended):Tmatrix3_extended	1145
64.4.280	subtract(Tmatrix3_extended,Tmatrix3_extended):Tmatrix3_extended	1146
64.4.281	subtract(Tmatrix3_single,single):Tmatrix3_single	1146
64.4.282	subtract(Tmatrix3_single,Tmatrix3_single):Tmatrix3_single	1146
64.4.283	subtract(Tmatrix4_double,Double):Tmatrix4_double	1146
64.4.284	subtract(Tmatrix4_double,Tmatrix4_double):Tmatrix4_double	1146
64.4.285	subtract(Tmatrix4_extended,extended):Tmatrix4_extended	1147
64.4.286	subtract(Tmatrix4_extended,Tmatrix4_extended):Tmatrix4_extended	1147
64.4.287	subtract(Tmatrix4_single,single):Tmatrix4_single	1147
64.4.288	subtract(Tmatrix4_single,Tmatrix4_single):Tmatrix4_single	1147
64.4.289	subtract(Tvector2_double,Double):Tvector2_double	1147
64.4.290	subtract(Tvector2_double,Tvector2_double):Tvector2_double	1148
64.4.291	subtract(Tvector2_extended,extended):Tvector2_extended	1148
64.4.292	subtract(Tvector2_extended,Tvector2_extended):Tvector2_extended	1148

64.4.293	subtract(Tvector2_single,single):Tvector2_single	1148
64.4.294	subtract(Tvector2_single,Tvector2_single):Tvector2_single	1148
64.4.295	subtract(Tvector3_double,Double):Tvector3_double	1149
64.4.296	subtract(Tvector3_double,Tvector3_double):Tvector3_double	1149
64.4.297	subtract(Tvector3_extended,extended):Tvector3_extended	1149
64.4.298	subtract(Tvector3_extended,Tvector3_extended):Tvector3_extended	1149
64.4.299	subtract(Tvector3_single,single):Tvector3_single	1149
64.4.300	subtract(Tvector3_single,Tvector3_single):Tvector3_single	1150
64.4.301	subtract(Tvector4_double,Double):Tvector4_double	1150
64.4.302	subtract(Tvector4_double,Tvector4_double):Tvector4_double	1150
64.4.303	subtract(Tvector4_extended,extended):Tvector4_extended	1150
64.4.304	subtract(Tvector4_extended,Tvector4_extended):Tvector4_extended	1150
64.4.305	subtract(Tvector4_single,single):Tvector4_single	1151
64.4.306	subtract(Tvector4_single,Tvector4_single):Tvector4_single	1151
64.4.307	symmetricaldifference(Tvector3_double,Tvector3_double):Tvector3_double	1151
64.4.308	symmetricaldifference(Tvector3_extended,Tvector3_extended):Tvector3_extended	1151
64.4.309	symmetricaldifference(Tvector3_single,Tvector3_single):Tvector3_single	1152
64.5	Tmatrix2_double	1152
64.5.1	Description	1152
64.5.2	Method overview	1152
64.5.3	Tmatrix2_double.init_zero	1152
64.5.4	Tmatrix2_double.init_identity	1152
64.5.5	Tmatrix2_double.init	1153
64.5.6	Tmatrix2_double.get_column	1153
64.5.7	Tmatrix2_double.get_row	1153
64.5.8	Tmatrix2_double.set_column	1153
64.5.9	Tmatrix2_double.set_row	1153
64.5.10	Tmatrix2_double.determinant	1153
64.5.11	Tmatrix2_double.inverse	1154
64.5.12	Tmatrix2_double.transpose	1154
64.6	Tmatrix2_extended	1154
64.6.1	Description	1154
64.6.2	Method overview	1154
64.6.3	Tmatrix2_extended.init_zero	1154
64.6.4	Tmatrix2_extended.init_identity	1155
64.6.5	Tmatrix2_extended.init	1155
64.6.6	Tmatrix2_extended.get_column	1155
64.6.7	Tmatrix2_extended.get_row	1155
64.6.8	Tmatrix2_extended.set_column	1155
64.6.9	Tmatrix2_extended.set_row	1155

64.6.10	Tmatrix2_extended.determinant	1156
64.6.11	Tmatrix2_extended.inverse	1156
64.6.12	Tmatrix2_extended.transpose	1156
64.7	Tmatrix2_single	1156
64.7.1	Description	1156
64.7.2	Method overview	1156
64.7.3	Tmatrix2_single.init_zero	1157
64.7.4	Tmatrix2_single.init_identity	1157
64.7.5	Tmatrix2_single.init	1157
64.7.6	Tmatrix2_single.get_column	1157
64.7.7	Tmatrix2_single.get_row	1157
64.7.8	Tmatrix2_single.set_column	1157
64.7.9	Tmatrix2_single.set_row	1158
64.7.10	Tmatrix2_single.determinant	1158
64.7.11	Tmatrix2_single.inverse	1158
64.7.12	Tmatrix2_single.transpose	1158
64.8	Tmatrix3_double	1158
64.8.1	Description	1158
64.8.2	Method overview	1159
64.8.3	Tmatrix3_double.init_zero	1159
64.8.4	Tmatrix3_double.init_identity	1159
64.8.5	Tmatrix3_double.init	1159
64.8.6	Tmatrix3_double.get_column	1159
64.8.7	Tmatrix3_double.get_row	1160
64.8.8	Tmatrix3_double.set_column	1160
64.8.9	Tmatrix3_double.set_row	1160
64.8.10	Tmatrix3_double.determinant	1160
64.8.11	Tmatrix3_double.inverse	1160
64.8.12	Tmatrix3_double.transpose	1160
64.9	Tmatrix3_extended	1161
64.9.1	Description	1161
64.9.2	Method overview	1161
64.9.3	Tmatrix3_extended.init_zero	1161
64.9.4	Tmatrix3_extended.init_identity	1161
64.9.5	Tmatrix3_extended.init	1161
64.9.6	Tmatrix3_extended.get_column	1162
64.9.7	Tmatrix3_extended.get_row	1162
64.9.8	Tmatrix3_extended.set_column	1162
64.9.9	Tmatrix3_extended.set_row	1162
64.9.10	Tmatrix3_extended.determinant	1162

64.9.11	Tmatrix3_extended.inverse	1162
64.9.12	Tmatrix3_extended.transpose	1163
64.10	Tmatrix3_single	1163
64.10.1	Description	1163
64.10.2	Method overview	1163
64.10.3	Tmatrix3_single.init_zero	1163
64.10.4	Tmatrix3_single.init_identity	1163
64.10.5	Tmatrix3_single.init	1164
64.10.6	Tmatrix3_single.get_column	1164
64.10.7	Tmatrix3_single.get_row	1164
64.10.8	Tmatrix3_single.set_column	1164
64.10.9	Tmatrix3_single.set_row	1164
64.10.10	Tmatrix3_single.determinant	1165
64.10.11	Tmatrix3_single.inverse	1165
64.10.12	Tmatrix3_single.transpose	1165
64.11	Tmatrix4_double	1165
64.11.1	Description	1165
64.11.2	Method overview	1165
64.11.3	Tmatrix4_double.init_zero	1166
64.11.4	Tmatrix4_double.init_identity	1166
64.11.5	Tmatrix4_double.init	1166
64.11.6	Tmatrix4_double.get_column	1166
64.11.7	Tmatrix4_double.get_row	1166
64.11.8	Tmatrix4_double.set_column	1167
64.11.9	Tmatrix4_double.set_row	1167
64.11.10	Tmatrix4_double.determinant	1167
64.11.11	Tmatrix4_double.inverse	1167
64.11.12	Tmatrix4_double.transpose	1167
64.12	Tmatrix4_extended	1168
64.12.1	Description	1168
64.12.2	Method overview	1168
64.12.3	Tmatrix4_extended.init_zero	1168
64.12.4	Tmatrix4_extended.init_identity	1168
64.12.5	Tmatrix4_extended.init	1168
64.12.6	Tmatrix4_extended.get_column	1169
64.12.7	Tmatrix4_extended.get_row	1169
64.12.8	Tmatrix4_extended.set_column	1169
64.12.9	Tmatrix4_extended.set_row	1169
64.12.10	Tmatrix4_extended.determinant	1169
64.12.11	Tmatrix4_extended.inverse	1170

64.12.12	Tmatrix4_extended.transpose	1170
64.13	Tmatrix4_single	1170
64.13.1	Description	1170
64.13.2	Method overview	1170
64.13.3	Tmatrix4_single.init_zero	1170
64.13.4	Tmatrix4_single.init_identity	1171
64.13.5	Tmatrix4_single.init	1171
64.13.6	Tmatrix4_single.get_column	1171
64.13.7	Tmatrix4_single.get_row	1171
64.13.8	Tmatrix4_single.set_column	1171
64.13.9	Tmatrix4_single.set_row	1172
64.13.10	Tmatrix4_single.determinant	1172
64.13.11	Tmatrix4_single.inverse	1172
64.13.12	Tmatrix4_single.transpose	1172
64.14	Tvector2_double	1172
64.14.1	Description	1172
64.14.2	Method overview	1173
64.14.3	Tvector2_double.init_zero	1173
64.14.4	Tvector2_double.init_one	1173
64.14.5	Tvector2_double.init	1173
64.14.6	Tvector2_double.length	1173
64.14.7	Tvector2_double.squared_length	1173
64.15	Tvector2_extended	1174
64.15.1	Description	1174
64.15.2	Method overview	1174
64.15.3	Tvector2_extended.init_zero	1174
64.15.4	Tvector2_extended.init_one	1174
64.15.5	Tvector2_extended.init	1174
64.15.6	Tvector2_extended.length	1174
64.15.7	Tvector2_extended.squared_length	1175
64.16	Tvector2_single	1175
64.16.1	Description	1175
64.16.2	Method overview	1175
64.16.3	Tvector2_single.init_zero	1175
64.16.4	Tvector2_single.init_one	1175
64.16.5	Tvector2_single.init	1175
64.16.6	Tvector2_single.length	1176
64.16.7	Tvector2_single.squared_length	1176
64.17	Tvector3_double	1176
64.17.1	Description	1176

64.17.2	Method overview	1176
64.17.3	Tvector3_double.init_zero	1176
64.17.4	Tvector3_double.init_one	1176
64.17.5	Tvector3_double.init	1177
64.17.6	Tvector3_double.length	1177
64.17.7	Tvector3_double.squared_length	1177
64.18	Tvector3_extended	1177
64.18.1	Description	1177
64.18.2	Method overview	1177
64.18.3	Tvector3_extended.init_zero	1177
64.18.4	Tvector3_extended.init_one	1178
64.18.5	Tvector3_extended.init	1178
64.18.6	Tvector3_extended.length	1178
64.18.7	Tvector3_extended.squared_length	1178
64.19	Tvector3_single	1178
64.19.1	Description	1178
64.19.2	Method overview	1178
64.19.3	Tvector3_single.init_zero	1179
64.19.4	Tvector3_single.init_one	1179
64.19.5	Tvector3_single.init	1179
64.19.6	Tvector3_single.length	1179
64.19.7	Tvector3_single.squared_length	1179
64.20	Tvector4_double	1179
64.20.1	Description	1179
64.20.2	Method overview	1180
64.20.3	Tvector4_double.init_zero	1180
64.20.4	Tvector4_double.init_one	1180
64.20.5	Tvector4_double.init	1180
64.20.6	Tvector4_double.length	1180
64.20.7	Tvector4_double.squared_length	1180
64.21	Tvector4_extended	1181
64.21.1	Description	1181
64.21.2	Method overview	1181
64.21.3	Tvector4_extended.init_zero	1181
64.21.4	Tvector4_extended.init_one	1181
64.21.5	Tvector4_extended.init	1181
64.21.6	Tvector4_extended.length	1181
64.21.7	Tvector4_extended.squared_length	1182
64.22	Tvector4_single	1182
64.22.1	Description	1182

64.22.2	Method overview	1182
64.22.3	Tvector4_single.init_zero	1182
64.22.4	Tvector4_single.init_one	1182
64.22.5	Tvector4_single.init	1182
64.22.6	Tvector4_single.length	1183
64.22.7	Tvector4_single.squared_length	1183
65	Reference for unit 'mmx'	1184
65.1	Used units	1184
65.2	Overview	1184
65.3	Constants, types and variables	1184
65.3.1	Constants	1184
65.3.2	Types	1185
65.4	Procedures and functions	1186
65.4.1	emms	1186
65.4.2	femms	1187
66	Reference for unit 'Mouse'	1188
66.1	Used units	1188
66.2	Overview	1188
66.3	Writing a custom mouse driver.	1188
66.4	Constants, types and variables	1190
66.4.1	Constants	1190
66.4.2	Types	1192
66.4.3	Variables	1192
66.5	Procedures and functions	1192
66.5.1	DetectMouse	1192
66.5.2	DoneMouse	1193
66.5.3	GetMouseButtons	1193
66.5.4	GetMouseDriver	1194
66.5.5	GetMouseEvent	1194
66.5.6	GetMouseX	1194
66.5.7	GetMouseY	1195
66.5.8	HideMouse	1195
66.5.9	InitMouse	1196
66.5.10	PollMouseEvent	1196
66.5.11	PutMouseEvent	1197
66.5.12	SetMouseDriver	1197
66.5.13	SetMouseXY	1197
66.5.14	ShowMouse	1198
66.6	TMouseDriver	1198

66.7	TMouseEvent	1199
67	Reference for unit 'Objects'	1200
67.1	Used units	1200
67.2	Overview	1200
67.3	Constants, types and variables	1200
67.3.1	Constants	1200
67.3.2	Types	1203
67.3.3	Variables	1205
67.4	Procedures and functions	1205
67.4.1	Abstract	1205
67.4.2	CallPointerConstructor	1206
67.4.3	CallPointerLocal	1206
67.4.4	CallPointerMethod	1206
67.4.5	CallPointerMethodLocal	1207
67.4.6	CallVoidConstructor	1207
67.4.7	CallVoidLocal	1207
67.4.8	CallVoidMethod	1208
67.4.9	CallVoidMethodLocal	1208
67.4.10	DisposeStr	1208
67.4.11	LongDiv	1208
67.4.12	LongMul	1209
67.4.13	NewStr	1209
67.4.14	RegisterObjects	1210
67.4.15	RegisterType	1210
67.4.16	SetStr	1212
67.5	LongRec	1212
67.6	PtrRec	1212
67.7	TStreamRec	1212
67.8	TStrIndexRec	1213
67.9	WordRec	1213
67.10	TBufStream	1213
67.10.1	Description	1213
67.10.2	Method overview	1214
67.10.3	TBufStream.Init	1214
67.10.4	TBufStream.Done	1214
67.10.5	TBufStream.Close	1215
67.10.6	TBufStream.Flush	1215
67.10.7	TBufStream.Truncate	1216
67.10.8	TBufStream.Seek	1216

67.10.9	TBufStream.Open	1216
67.10.10	TBufStream.Read	1217
67.10.11	TBufStream.Write	1217
67.11	TCollection	1217
67.11.1	Description	1217
67.11.2	Method overview	1218
67.11.3	TCollection.Init	1218
67.11.4	TCollection.Load	1218
67.11.5	TCollection.Done	1219
67.11.6	TCollection.At	1219
67.11.7	TCollection.IndexOf	1220
67.11.8	TCollection.GetItem	1221
67.11.9	TCollection.LastThat	1221
67.11.10	TCollection.FirstThat	1222
67.11.11	TCollection.Pack	1223
67.11.12	TCollection.FreeAll	1224
67.11.13	TCollection.DeleteAll	1225
67.11.14	TCollection.Free	1225
67.11.15	TCollection.Insert	1226
67.11.16	TCollection.Delete	1226
67.11.17	TCollection.AtFree	1227
67.11.18	TCollection.FreeItem	1228
67.11.19	TCollection.AtDelete	1228
67.11.20	TCollection.ForEach	1229
67.11.21	TCollection.SetLimit	1230
67.11.22	TCollection.Error	1230
67.11.23	TCollection.AtPut	1230
67.11.24	TCollection.AtInsert	1231
67.11.25	TCollection.Store	1232
67.11.26	TCollection.PutItem	1232
67.12	TDosStream	1232
67.12.1	Description	1232
67.12.2	Method overview	1232
67.12.3	TDosStream.Init	1233
67.12.4	TDosStream.Done	1233
67.12.5	TDosStream.Close	1233
67.12.6	TDosStream.Truncate	1234
67.12.7	TDosStream.Seek	1234
67.12.8	TDosStream.Open	1235
67.12.9	TDosStream.Read	1236

67.12.10	TDosStream.Write	1236
67.13	TMemoryStream	1237
67.13.1	Description	1237
67.13.2	Method overview	1237
67.13.3	TMemoryStream.Init	1237
67.13.4	TMemoryStream.Done	1237
67.13.5	TMemoryStream.Truncate	1238
67.13.6	TMemoryStream.Read	1238
67.13.7	TMemoryStream.Write	1239
67.14	TObject	1239
67.14.1	Description	1239
67.14.2	Method overview	1239
67.14.3	TObject.Init	1239
67.14.4	TObject.Free	1240
67.14.5	TObject.Is_Object	1240
67.14.6	TObject.Done	1240
67.15	TPoint	1241
67.15.1	Description	1241
67.16	TRect	1241
67.16.1	Description	1241
67.16.2	Method overview	1241
67.16.3	TRect.Empty	1241
67.16.4	TRect.Equals	1242
67.16.5	TRect.Contains	1243
67.16.6	TRect.Copy	1243
67.16.7	TRect.Union	1244
67.16.8	TRect.Intersect	1244
67.16.9	TRect.Move	1245
67.16.10	TRect.Grow	1246
67.16.11	TRect.Assign	1246
67.17	TResourceCollection	1247
67.17.1	Description	1247
67.17.2	Method overview	1247
67.17.3	TResourceCollection.KeyOf	1247
67.17.4	TResourceCollection.GetItem	1248
67.17.5	TResourceCollection.FreeItem	1248
67.17.6	TResourceCollection.PutItem	1248
67.18	TResourceFile	1248
67.18.1	Description	1248
67.18.2	Method overview	1249

67.18.3	TResourceFile.Init	1249
67.18.4	TResourceFile.Done	1249
67.18.5	TResourceFile.Count	1249
67.18.6	TResourceFile.KeyAt	1250
67.18.7	TResourceFile.Get	1250
67.18.8	TResourceFile.SwitchTo	1250
67.18.9	TResourceFile.Flush	1250
67.18.10	TResourceFile.Delete	1251
67.18.11	TResourceFile.Put	1251
67.19	TSortedCollection	1251
67.19.1	Description	1251
67.19.2	Method overview	1252
67.19.3	TSortedCollection.Init	1252
67.19.4	TSortedCollection.Load	1252
67.19.5	TSortedCollection.KeyOf	1252
67.19.6	TSortedCollection.IndexOf	1253
67.19.7	TSortedCollection.Compare	1253
67.19.8	TSortedCollection.Search	1254
67.19.9	TSortedCollection.Insert	1255
67.19.10	TSortedCollection.Store	1256
67.20	TStrCollection	1257
67.20.1	Description	1257
67.20.2	Method overview	1257
67.20.3	TStrCollection.Compare	1257
67.20.4	TStrCollection.GetItem	1258
67.20.5	TStrCollection.FreeItem	1258
67.20.6	TStrCollection.PutItem	1258
67.21	TStream	1259
67.21.1	Description	1259
67.21.2	Method overview	1259
67.21.3	TStream.Init	1259
67.21.4	TStream.Get	1259
67.21.5	TStream.StrRead	1260
67.21.6	TStream.GetPos	1261
67.21.7	TStream.GetSize	1261
67.21.8	TStream.ReadStr	1262
67.21.9	TStream.Open	1263
67.21.10	TStream.Close	1263
67.21.11	TStream.Reset	1263
67.21.12	TStream.Flush	1264

67.21.13	TStream.Truncate	1264
67.21.14	TStream.Put	1264
67.21.15	TStream.StrWrite	1265
67.21.16	TStream.WriteStr	1265
67.21.17	TStream.Seek	1265
67.21.18	TStream.Error	1265
67.21.19	TStream.Read	1266
67.21.20	TStream.Write	1266
67.21.21	TStream.CopyFrom	1267
67.22	TStringCollection	1267
67.22.1	Description	1267
67.22.2	Method overview	1268
67.22.3	TStringCollection.GetItem	1268
67.22.4	TStringCollection.Compare	1268
67.22.5	TStringCollection.FreeItem	1269
67.22.6	TStringCollection.PutItem	1269
67.23	TStringList	1269
67.23.1	Description	1269
67.23.2	Method overview	1270
67.23.3	TStringList.Load	1270
67.23.4	TStringList.Done	1270
67.23.5	TStringList.Get	1270
67.24	TStrListMaker	1271
67.24.1	Description	1271
67.24.2	Method overview	1271
67.24.3	TStrListMaker.Init	1271
67.24.4	TStrListMaker.Done	1271
67.24.5	TStrListMaker.Put	1271
67.24.6	TStrListMaker.Store	1272
67.25	TUnSortedStrCollection	1272
67.25.1	Description	1272
67.25.2	Method overview	1272
67.25.3	TUnSortedStrCollection.Insert	1272
68	Reference for unit 'objpas'	1274
68.1	Used units	1274
68.2	Overview	1274
68.3	Constants, types and variables	1274
68.3.1	Constants	1274
68.3.2	Types	1274

69	Reference for unit 'ports'	1276
69.1	Used units	1276
69.2	Overview	1276
69.3	Constants, types and variables	1276
69.3.1	Variables	1276
69.4	tport	1277
69.4.1	Description	1277
69.4.2	Property overview	1277
69.4.3	tport.pp	1277
69.5	tportl	1278
69.5.1	Description	1278
69.5.2	Property overview	1278
69.5.3	tportl.pp	1278
69.6	tportw	1278
69.6.1	Description	1278
69.6.2	Property overview	1278
69.6.3	tportw.pp	1278
70	Reference for unit 'printer'	1279
70.1	Used units	1279
70.2	Overview	1279
70.3	Constants, types and variables	1279
70.3.1	Variables	1279
70.4	Procedures and functions	1279
70.4.1	AssignLst	1279
70.4.2	InitPrinter	1280
70.4.3	IsLstAvailable	1280
71	Reference for unit 'sharemem'	1281
71.1	Used units	1281
71.2	Overview	1281
72	Reference for unit 'Sockets'	1282
72.1	Used units	1282
72.2	Overview	1282
73	Reference for unit 'Strings'	1283
73.1	Used units	1283
73.2	Overview	1283
73.3	Procedures and functions	1283
73.3.1	stralloc	1283

73.3.2	strcat	1283
73.3.3	strcmp	1284
73.3.4	strcpy	1284
73.3.5	strdispose	1285
73.3.6	strecopy	1286
73.3.7	strend	1286
73.3.8	stricmp	1287
73.3.9	stripos	1287
73.3.10	striscan	1288
73.3.11	strlcat	1288
73.3.12	strlcomp	1289
73.3.13	strlcopy	1289
73.3.14	strlen	1290
73.3.15	strlicomp	1290
73.3.16	strlower	1291
73.3.17	strmove	1291
73.3.18	strnew	1292
73.3.19	strupas	1293
73.3.20	strpcopy	1293
73.3.21	strpos	1294
73.3.22	striscan	1294
73.3.23	strscan	1295
73.3.24	strscan	1295
73.3.25	strupper	1295
74	Reference for unit 'StrUtils'	1296
74.1	Used units	1296
74.2	Constants, types and variables	1296
74.2.1	Resource strings	1296
74.2.2	Constants	1296
74.2.3	Types	1297
74.3	Procedures and functions	1299
74.3.1	AddChar	1299
74.3.2	AddCharR	1299
74.3.3	AnsiContainsStr	1299
74.3.4	AnsiContainsText	1300
74.3.5	AnsiEndsStr	1300
74.3.6	AnsiEndsText	1300
74.3.7	AnsiIndexStr	1301
74.3.8	AnsiIndexText	1301

74.3.9	AnsiLeftStr	1301
74.3.10	AnsiMatchStr	1302
74.3.11	AnsiMatchText	1302
74.3.12	AnsiMidStr	1302
74.3.13	AnsiProperCase	1302
74.3.14	AnsiReplaceStr	1303
74.3.15	AnsiReplaceText	1303
74.3.16	AnsiResemblesText	1303
74.3.17	AnsiReverseString	1304
74.3.18	AnsiRightStr	1304
74.3.19	AnsiStartsStr	1304
74.3.20	AnsiStartsText	1304
74.3.21	BinToHex	1305
74.3.22	ContainsStr	1305
74.3.23	ContainsText	1306
74.3.24	Copy2Space	1306
74.3.25	Copy2SpaceDel	1306
74.3.26	Copy2Symb	1306
74.3.27	Copy2SymbDel	1307
74.3.28	Dec2Numb	1307
74.3.29	DecodeSoundexInt	1307
74.3.30	DecodeSoundexWord	1308
74.3.31	DelChars	1308
74.3.32	DelSpace	1308
74.3.33	DelSpace1	1308
74.3.34	DupeString	1309
74.3.35	EndsStr	1309
74.3.36	EndsText	1309
74.3.37	ExtractDelimited	1310
74.3.38	ExtractSubstr	1310
74.3.39	ExtractWord	1310
74.3.40	ExtractWordPos	1311
74.3.41	FindMatchesBoyerMooreCaseInsensitive	1311
74.3.42	FindMatchesBoyerMooreCaseSensitive	1312
74.3.43	FindPart	1312
74.3.44	GetCmdLineArg	1312
74.3.45	Hex2Dec	1313
74.3.46	Hex2Dec64	1313
74.3.47	HexToBin	1313
74.3.48	IfThen	1314

74.3.49	in(Ansistring,):Boolean	1314
74.3.50	in(UnicodeString,):Boolean	1314
74.3.51	IndexStr	1315
74.3.52	IndexText	1315
74.3.53	IntToBin	1315
74.3.54	IntToRoman	1316
74.3.55	IsEmptyStr	1316
74.3.56	IsWild	1316
74.3.57	IsWordPresent	1317
74.3.58	LeftBStr	1317
74.3.59	LeftStr	1317
74.3.60	MatchStr	1318
74.3.61	MatchText	1318
74.3.62	MidBStr	1318
74.3.63	MidStr	1318
74.3.64	NaturalCompareText	1319
74.3.65	NPos	1319
74.3.66	Numb2Dec	1320
74.3.67	Numb2USA	1320
74.3.68	PadCenter	1320
74.3.69	PadLeft	1320
74.3.70	PadRight	1321
74.3.71	PosEx	1321
74.3.72	PosSet	1321
74.3.73	PosSetEx	1322
74.3.74	RandomFrom	1322
74.3.75	RemoveLeadingChars	1322
74.3.76	RemovePadChars	1322
74.3.77	RemoveTrailingChars	1323
74.3.78	ReplaceStr	1323
74.3.79	ReplaceText	1323
74.3.80	ResemblesText	1324
74.3.81	ReverseString	1324
74.3.82	RightBStr	1324
74.3.83	RightStr	1324
74.3.84	RomanToInt	1325
74.3.85	RomanToIntDef	1325
74.3.86	RPos	1325
74.3.87	RPosEx	1326
74.3.88	SearchBuf	1326

74.3.89	Soundex	1327
74.3.90	SoundexCompare	1327
74.3.91	SoundexInt	1327
74.3.92	SoundexProc	1328
74.3.93	SoundexSimilar	1328
74.3.94	SoundexWord	1328
74.3.95	SplitCommandLine	1329
74.3.96	SplitString	1329
74.3.97	StartsStr	1329
74.3.98	StartsText	1330
74.3.99	StringReplace	1330
74.3.100	StringsReplace	1330
74.3.101	StuffString	1331
74.3.102	Tab2Space	1331
74.3.103	TrimLeftSet	1331
74.3.104	TrimRightSet	1332
74.3.105	TrimSet	1332
74.3.106	TryRomanToInt	1332
74.3.107	WordCount	1332
74.3.108	WordPosition	1333
74.3.109	XorDecode	1333
74.3.110	XorEncode	1333
74.3.111	XorString	1334
75	Reference for unit 'System'	1335
75.1	Overview	1335
75.2	A list of managed types.	1335
75.3	Unicode and codepage support.	1336
75.4	Miscellaneous functions.	1337
75.5	Operating System functions.	1338
75.6	String handling.	1338
75.7	Mathematical routines.	1338
75.8	Memory management functions.	1339
75.9	File handling functions.	1340
75.10	Run-Time Error behaviour.	1341
75.11	Constants, types and variables	1341
75.11.1	Constants	1341
75.11.2	Types	1369
75.11.3	Variables	1413
75.12	Procedures and functions	1418

75.12.1	Abs	1418
75.12.2	AbstractError	1419
75.12.3	AcquireExceptionObject	1419
75.12.4	add(variant,variant):variant	1419
75.12.5	AddExitProc	1419
75.12.6	Addr	1420
75.12.7	Align	1420
75.12.8	AllocMem	1421
75.12.9	AnsiToUtf8	1421
75.12.10	Append	1421
75.12.11	ArcTan	1422
75.12.12	ArrayStringToPPchar	1422
75.12.13	Assert	1423
75.12.14	Assign	1423
75.12.15	assign(Comp):olevariant	1424
75.12.16	assign(Comp):variant	1424
75.12.17	assign(extended):olevariant	1424
75.12.18	assign(extended):variant	1424
75.12.19	assign(NativeInt):variant	1424
75.12.20	assign(NativeUInt):variant	1425
75.12.21	assign(olevariant):Comp	1425
75.12.22	assign(olevariant):extended	1425
75.12.23	assign(olevariant):Real	1425
75.12.24	assign(olevariant):single	1425
75.12.25	assign(olevariant):UnicodeString	1425
75.12.26	assign(Real):olevariant	1425
75.12.27	assign(Real):variant	1426
75.12.28	assign(Real48):extended	1426
75.12.29	assign(single):olevariant	1426
75.12.30	assign(single):variant	1426
75.12.31	assign(UCS4String):variant	1426
75.12.32	assign(UnicodeString):olevariant	1426
75.12.33	assign(UnicodeString):variant	1427
75.12.34	assign(UTF8String):variant	1427
75.12.35	assign(variant):Comp	1427
75.12.36	assign(variant):extended	1427
75.12.37	assign(variant):NativeInt	1427
75.12.38	assign(variant):NativeUInt	1427
75.12.39	assign(variant):Real	1427
75.12.40	assign(variant):single	1428

75.12.41	assign(variant):unicodestring	1428
75.12.42	assign(variant):UTF8String	1428
75.12.43	Assigned	1428
75.12.44	BasicEventCreate	1429
75.12.45	BasicEventDestroy	1429
75.12.46	BasicEventResetEvent	1429
75.12.47	BasicEventSetEvent	1429
75.12.48	BasicEventWaitFor	1429
75.12.49	BeginThread	1430
75.12.50	BEtoN	1430
75.12.51	BinStr	1430
75.12.52	BlockRead	1431
75.12.53	BlockWrite	1432
75.12.54	Break	1432
75.12.55	BsfByte	1434
75.12.56	BsfDWord	1434
75.12.57	BsfQWord	1434
75.12.58	BsfWord	1435
75.12.59	BsrByte	1435
75.12.60	BsrDWord	1435
75.12.61	BsrQWord	1436
75.12.62	BsrWord	1436
75.12.63	CaptureBacktrace	1436
75.12.64	ChDir	1436
75.12.65	Chr	1437
75.12.66	Close	1437
75.12.67	CloseThread	1438
75.12.68	CompareByte	1438
75.12.69	CompareChar	1439
75.12.70	CompareChar0	1441
75.12.71	CompareDWord	1441
75.12.72	CompareWord	1442
75.12.73	Concat	1443
75.12.74	Continue	1444
75.12.75	Copy	1445
75.12.76	CopyArray	1446
75.12.77	Cos	1446
75.12.78	CSeg	1447
75.12.79	Dec	1447
75.12.80	Default	1448

75.12.81 DefaultAnsi2UnicodeMove	1448
75.12.82 DefaultAnsi2WideMove	1449
75.12.83 DefaultUnicode2AnsiMove	1449
75.12.84 Delete	1449
75.12.85 Dispose	1450
75.12.86 divide(variant,variant):variant	1451
75.12.87 DoneCriticalSection	1451
75.12.88 DoneThread	1452
75.12.89 DSeg	1452
75.12.90 DumpExceptionBacktrace	1452
75.12.91 Dump_Stack	1453
75.12.92 DynArrayBounds	1453
75.12.93 DynArrayClear	1453
75.12.94 DynArrayDim	1453
75.12.95 DynArrayIndex	1454
75.12.96 DynArraySetLength	1454
75.12.97 DynArraySize	1454
75.12.98 EmptyMethod	1454
75.12.99 EndThread	1455
75.12.100EnterCriticalSection	1455
75.12.101EnumResourceLanguages	1455
75.12.102EnumResourceNames	1456
75.12.103EnumResourceTypes	1456
75.12.104EOF	1456
75.12.105EOLn	1457
75.12.106equal(variant,variant):Boolean	1458
75.12.107Erase	1458
75.12.108Error	1459
75.12.109Exclude	1459
75.12.110Exit	1460
75.12.111Exp	1462
75.12.112Fail	1462
75.12.113FilePos	1463
75.12.114FileSize	1464
75.12.115FillByte	1465
75.12.116FillChar	1465
75.12.117FillDWord	1466
75.12.118FillWord	1466
75.12.119Finalize	1467
75.12.120FinalizeArray	1467

75.12.121FindResource	1468
75.12.122FindResourceEx	1468
75.12.123float_raise	1469
75.12.124Flush	1469
75.12.125FlushThread	1470
75.12.126FMADouble	1470
75.12.127FMAExtended	1470
75.12.128FMASingle	1470
75.12.129FPower10	1470
75.12.130Frac	1471
75.12.131FreeLibrary	1471
75.12.132Freemem	1471
75.12.133Freememory	1472
75.12.134FreeResource	1472
75.12.135Get8087CW	1472
75.12.136GetCPUCount	1472
75.12.137GetCurrentThreadId	1473
75.12.138GetDir	1473
75.12.139GetDynLibsManager	1474
75.12.140GetFPCHeapStatus	1474
75.12.141GetHeapStatus	1474
75.12.142GetLoadErrorStr	1474
75.12.143GetMem	1474
75.12.144GetMemory	1475
75.12.145GetMemoryManager	1475
75.12.146GetMXCSR	1475
75.12.147GetProcAddress	1475
75.12.148GetProcedureAddress	1476
75.12.149GetProcessID	1476
75.12.150GetResourceManager	1476
75.12.151GetSSECSR	1476
75.12.152GetTextAutoFlush	1477
75.12.153GetTextCodePage	1477
75.12.154GetThreadID	1477
75.12.155GetThreadManager	1477
75.12.156GetTypeKind	1477
75.12.157GetUnicodeStringManager	1478
75.12.158GetVariantManager	1478
75.12.159GetWideStringManager	1478
75.12.160get_caller_addr	1479

75.12.161get_caller_frame	1479
75.12.162get_caller_stackinfo	1479
75.12.163get_cmdline	1479
75.12.164get_frame	1480
75.12.165Get_pc_addr	1480
75.12.166greaterthan(variant,variant):Boolean	1480
75.12.167greaterthanorequal(variant,variant):Boolean	1480
75.12.168Halt	1481
75.12.169HexStr	1481
75.12.170Hi	1482
75.12.171High	1483
75.12.172HINSTANCE	1484
75.12.173Inc	1484
75.12.174Include	1485
75.12.175IndexByte	1485
75.12.176IndexChar	1486
75.12.177IndexChar0	1487
75.12.178IndexDWord	1487
75.12.179IndexQWord	1488
75.12.180Indexword	1488
75.12.181InitCriticalSection	1489
75.12.182Initialize	1489
75.12.183InitializeArray	1492
75.12.184InitThread	1492
75.12.185InitThreadVars	1492
75.12.186Insert	1493
75.12.187Int	1493
75.12.188intdivide(variant,variant):variant	1494
75.12.189InterlockedCompareExchange	1494
75.12.190InterlockedCompareExchange64	1495
75.12.191InterlockedCompareExchangePointer	1495
75.12.192InterlockedDecrement	1495
75.12.193InterlockedDecrement64	1495
75.12.194InterlockedExchange	1495
75.12.195InterlockedExchange64	1496
75.12.196InterlockedExchangeAdd	1496
75.12.197InterlockedExchangeAdd64	1496
75.12.198InterlockedIncrement	1497
75.12.199InterlockedIncrement64	1497
75.12.200IOResult	1497

75.12.201IsDynArrayRectangular	1498
75.12.202IsMemoryManagerSet	1499
75.12.203Is_IntResource	1499
75.12.204KillThread	1499
75.12.205LazyInitThreading	1499
75.12.206LeaveCriticalSection	1499
75.12.207leftshift(variant,variant):variant	1500
75.12.208Length	1500
75.12.209lessthan(variant,variant):Boolean	1501
75.12.210lessthanorequal(variant,variant):Boolean	1501
75.12.211LEtoN	1501
75.12.212Ln	1502
75.12.213Lo	1502
75.12.214LoadLibrary	1503
75.12.215LoadResource	1503
75.12.216LockResource	1504
75.12.217logicaland(variant,variant):variant	1504
75.12.218logicalnot(variant):variant	1504
75.12.219logicalor(variant,variant):variant	1505
75.12.220logicalxor(variant,variant):variant	1505
75.12.221longjmp	1505
75.12.222Low	1506
75.12.223LowerCase	1506
75.12.224MakeLangID	1507
75.12.225MemSize	1507
75.12.226MkDir	1507
75.12.227modulus(variant,variant):variant	1507
75.12.228Move	1508
75.12.229MoveChar0	1508
75.12.230multiply(variant,variant):variant	1509
75.12.231negative(variant):variant	1509
75.12.232New	1509
75.12.233NtoBE	1510
75.12.234NtoLE	1510
75.12.235Null	1510
75.12.236OctStr	1511
75.12.237Odd	1511
75.12.238Ofs	1512
75.12.239Ord	1512
75.12.240Pack	1513

75.12.241ParamCount	1513
75.12.242ParamStr	1514
75.12.243Pi	1514
75.12.244PopCnt	1515
75.12.245Pos	1515
75.12.246power(variant,variant):variant	1516
75.12.247Pred	1517
75.12.248Prefetch	1517
75.12.249Ptr	1517
75.12.250RaiseList	1518
75.12.251Random	1518
75.12.252Randomize	1519
75.12.253Read	1519
75.12.254ReadBarrier	1520
75.12.255ReadDependencyBarrier	1520
75.12.256ReadLn	1521
75.12.257ReadStr	1521
75.12.258ReadWriteBarrier	1522
75.12.259Real2Double	1522
75.12.260ReAllocMem	1523
75.12.261ReAllocMemory	1523
75.12.262RegisterLazyInitThreadingProc	1523
75.12.263ReleaseExceptionObject	1523
75.12.264Rename	1524
75.12.265Reset	1524
75.12.266ResumeThread	1525
75.12.267Rewrite	1525
75.12.268rightshift(variant,variant):variant	1526
75.12.269Rmdir	1526
75.12.270RolByte	1527
75.12.271RolDWord	1527
75.12.272RolQWord	1528
75.12.273RolWord	1528
75.12.274RorByte	1528
75.12.275RorDWord	1529
75.12.276RorQWord	1529
75.12.277RorWord	1529
75.12.278Round	1529
75.12.279RTLEventCreate	1531
75.12.280RTLEventDestroy	1531

75.12.281RTLEventResetEvent	1531
75.12.282RTLEventSetEvent	1532
75.12.283RTLEventWaitFor	1532
75.12.284RunError	1532
75.12.285SafeLoadLibrary	1533
75.12.286SarInt64	1533
75.12.287SarLongint	1533
75.12.288SarShortint	1534
75.12.289SarSmallint	1534
75.12.290Seek	1534
75.12.291SeekEOF	1535
75.12.292SeekEOLn	1536
75.12.293Seg	1536
75.12.294Set8087CW	1537
75.12.295SetCodePage	1537
75.12.296SetDynLibsManager	1537
75.12.297Setjmp	1537
75.12.298SetLength	1538
75.12.299SetMemoryManager	1539
75.12.300SetMultiByteConversionCodePage	1540
75.12.301SetMultiByteFileSystemCodePage	1540
75.12.302SetMultiByteRTLFileSystemCodePage	1540
75.12.303SetMXCSR	1541
75.12.304SetResourceManager	1541
75.12.305SetSSECSR	1541
75.12.306SetString	1541
75.12.307SetTextAutoFlush	1542
75.12.308SetTextBuf	1542
75.12.309SetTextCodePage	1543
75.12.310SetTextLineEnding	1543
75.12.311SetThreadDebugName	1543
75.12.312SetThreadManager	1544
75.12.313SetUnicodeStringManager	1544
75.12.314SetVariantManager	1544
75.12.315SetWideStringManager	1545
75.12.316ShortCompareText	1545
75.12.317Sin	1545
75.12.318SizeOf	1546
75.12.319SizeofResource	1546
75.12.320Slice	1547

75.12.321Space	1547
75.12.322SPtr	1547
75.12.323Sqr	1548
75.12.324Sqrt	1548
75.12.325SSeg	1549
75.12.326StackTop	1549
75.12.327Str	1549
75.12.328StringCodePage	1551
75.12.329StringElementSize	1551
75.12.330StringOfChar	1552
75.12.331StringRefCount	1552
75.12.332StringToPPChar	1552
75.12.333StringToUnicodeChar	1553
75.12.334StringToWideChar	1553
75.12.335StrLen	1554
75.12.336StrPas	1554
75.12.337subtract(variant,variant):variant	1554
75.12.338Succ	1554
75.12.339SuspendThread	1555
75.12.340Swap	1555
75.12.341SwapEndian	1556
75.12.342SysAllocMem	1556
75.12.343SysAssert	1556
75.12.344SysBacktraceStr	1556
75.12.345SysFlushStdIO	1557
75.12.346SysFreemem	1557
75.12.347SysFreememSize	1557
75.12.348SysGetFPCHeapStatus	1557
75.12.349SysGetHeapStatus	1558
75.12.350SysGetmem	1558
75.12.351SysInitExceptions	1558
75.12.352SysInitFPU	1558
75.12.353SysInitStdIO	1558
75.12.354SysMemSize	1559
75.12.355SysReAllocMem	1559
75.12.356SysResetFPU	1559
75.12.357SysSetCtrlBreakHandler	1559
75.12.358SysTryResizeMem	1559
75.12.359ThreadGetPriority	1560
75.12.360ThreadSetPriority	1560

75.12.361ThreadSwitch	1560
75.12.362ToSingleByteFileSystemEncodedFileName	1560
75.12.363Trunc	1561
75.12.364Truncate	1562
75.12.365TryEnterCriticalSection	1563
75.12.366TypeInfo	1563
75.12.367TypeOf	1563
75.12.368UCS4StringToUnicodeString	1564
75.12.369UCS4StringToWideString	1564
75.12.370Unassigned	1564
75.12.371UnicodeCharLenToString	1564
75.12.372UnicodeCharLenToStrVar	1564
75.12.373UnicodeCharToString	1565
75.12.374UnicodeCharToStrVar	1565
75.12.375UnicodeStringToUCS4String	1565
75.12.376UnicodeToUtf8	1566
75.12.377UniqueString	1566
75.12.378UnloadLibrary	1566
75.12.379UnlockResource	1567
75.12.380UnPack	1567
75.12.381UpCase	1567
75.12.382Utf8CodePointLen	1568
75.12.383UTF8Decode	1568
75.12.384UTF8Encode	1569
75.12.385Utf8ToAnsi	1569
75.12.386UTF8ToString	1569
75.12.387Utf8ToUnicode	1570
75.12.388Val	1570
75.12.389VarArrayGet	1571
75.12.390VarArrayPut	1571
75.12.391VarArrayRedim	1572
75.12.392VarCast	1572
75.12.393WaitForThreadTerminate	1572
75.12.394WideCharLenToString	1573
75.12.395WideCharLenToStrVar	1573
75.12.396WideCharToString	1573
75.12.397WideCharToStrVar	1574
75.12.398WideStringToUCS4String	1574
75.12.399Write	1574
75.12.400WriteBarrier	1575

75.12.401WriteLn	1575
75.12.402WriteStr	1576
75.13 TDoubleRec	1577
75.13.1 Method overview	1578
75.13.2 Property overview	1578
75.13.3 TDoubleRec.Mantissa	1578
75.13.4 TDoubleRec.Fraction	1578
75.13.5 TDoubleRec.Exponent	1579
75.13.6 TDoubleRec.SpecialType	1579
75.13.7 TDoubleRec.BuildUp	1579
75.13.8 TDoubleRec.Sign	1579
75.13.9 TDoubleRec.Exp	1579
75.13.10 TDoubleRec.Frac	1580
75.14 TExtended80Rec	1580
75.14.1 Method overview	1581
75.14.2 Property overview	1581
75.14.3 TExtended80Rec.Mantissa	1581
75.14.4 TExtended80Rec.Fraction	1581
75.14.5 TExtended80Rec.Exponent	1582
75.14.6 TExtended80Rec.SpecialType	1582
75.14.7 TExtended80Rec.BuildUp	1582
75.14.8 TExtended80Rec.Sign	1582
75.14.9 TExtended80Rec.Exp	1582
75.15 TGuid	1583
75.15.1 Method overview	1583
75.15.2 TGuid.equal(TGUID,TGUID):Boolean	1584
75.15.3 TGuid.notequal(TGUID,TGUID):Boolean	1584
75.15.4 TGuid.Empty	1584
75.15.5 TGuid.Create	1584
75.15.6 TGuid.IsEmpty	1584
75.16 tinterfaceentry	1584
75.16.1 Property overview	1585
75.16.2 tinterfaceentry.IID	1585
75.16.3 tinterfaceentry.IIDStr	1585
75.17 TSingleRec	1585
75.17.1 Method overview	1586
75.17.2 Property overview	1587
75.17.3 TSingleRec.Mantissa	1587
75.17.4 TSingleRec.Fraction	1587
75.17.5 TSingleRec.Exponent	1587

75.17.6	TSingleRec.SpecialType	1587
75.17.7	TSingleRec.BuildUp	1588
75.17.8	TSingleRec.Sign	1588
75.17.9	TSingleRec.Exp	1588
75.17.10	TSingleRec.Frac	1588
75.18	TVmt	1588
75.18.1	Property overview	1589
75.18.2	TVmt.vParent	1589
75.19	IDispatch	1590
75.19.1	Description	1590
75.19.2	Method overview	1590
75.19.3	IDispatch.GetTypeInfoCount	1590
75.19.4	IDispatch.GetTypeInfo	1590
75.19.5	IDispatch.GetIDsOfNames	1590
75.19.6	IDispatch.Invoke	1590
75.20	IEnumerable	1591
75.20.1	Description	1591
75.20.2	Method overview	1591
75.20.3	IEnumerable.GetEnumerator	1591
75.21	IEnumerator	1591
75.21.1	Description	1591
75.21.2	Method overview	1592
75.21.3	Property overview	1592
75.21.4	IEnumerator.GetCurrent	1592
75.21.5	IEnumerator.MoveNext	1592
75.21.6	IEnumerator.Reset	1593
75.21.7	IEnumerator.Current	1593
75.22	IInvokable	1593
75.22.1	Description	1593
75.23	IUnknown	1593
75.23.1	Description	1593
75.23.2	Method overview	1593
75.23.3	IUnknown.QueryInterface	1594
75.23.4	IUnknown._AddRef	1594
75.23.5	IUnknown._Release	1594
75.24	TAggregatedObject	1594
75.24.1	Description	1594
75.24.2	Method overview	1594
75.24.3	Property overview	1594
75.24.4	TAggregatedObject.Create	1595

75.24.5	TAggregatedObject.Controller	1595
75.25	TContainedObject	1595
75.25.1	Description	1595
75.25.2	Interfaces overview	1595
75.26	TInterfacedObject	1595
75.26.1	Description	1595
75.26.2	Interfaces overview	1596
75.26.3	Method overview	1596
75.26.4	Property overview	1596
75.26.5	TInterfacedObject.destroy	1596
75.26.6	TInterfacedObject.AfterConstruction	1596
75.26.7	TInterfacedObject.BeforeDestruction	1596
75.26.8	TInterfacedObject.NewInstance	1597
75.26.9	TInterfacedObject.RefCount	1597
75.27	TObject	1597
75.27.1	Description	1597
75.27.2	Method overview	1598
75.27.3	TObject.Create	1598
75.27.4	TObject.Destroy	1599
75.27.5	TObject.newinstance	1599
75.27.6	TObject.FreeInstance	1599
75.27.7	TObject.SafeCallException	1600
75.27.8	TObject.DefaultHandler	1600
75.27.9	TObject.Free	1600
75.27.10	TObject.InitInstance	1600
75.27.11	TObject.CleanupInstance	1601
75.27.12	TObject.ClassType	1601
75.27.13	TObject.ClassInfo	1601
75.27.14	TObject.ClassName	1601
75.27.15	TObject.ClassNameIs	1602
75.27.16	TObject.ClassParent	1602
75.27.17	TObject.InstanceSize	1602
75.27.18	TObject.InheritsFrom	1602
75.27.19	TObject.StringMessageTable	1603
75.27.20	TObject.MethodAddress	1603
75.27.21	TObject.MethodName	1603
75.27.22	TObject.FieldAddress	1603
75.27.23	TObject.AfterConstruction	1604
75.27.24	TObject.BeforeDestruction	1604
75.27.25	TObject.DefaultHandlerStr	1604

75.27.26	TObject.Dispatch	1604
75.27.27	TObject.DispatchStr	1605
75.27.28	TObject.GetInterface	1605
75.27.29	TObject.GetInterfaceByStr	1605
75.27.30	TObject.GetInterfaceWeak	1606
75.27.31	TObject.GetInterfaceEntry	1606
75.27.32	TObject.GetInterfaceEntryByStr	1606
75.27.33	TObject.GetInterfaceTable	1606
75.27.34	TObject.UnitName	1607
75.27.35	TObject.QualifiedClassName	1607
75.27.36	TObject.Equals	1607
75.27.37	TObject.GetHashCode	1607
75.27.38	TObject.ToString	1608
76	Reference for unit 'sysutils'	1609
76.1	Used units	1609
76.2	Overview	1609
76.3	Type Helpers for basic types.	1609
76.4	Localization support.	1610
76.5	Unicode and codepage awareness.	1610
76.6	Miscellaneous conversion routines.	1611
76.7	Date/time routines.	1612
76.8	File Name handling routines.	1612
76.9	File input/output routines.	1613
76.10	PChar related functions.	1614
76.11	Date and time formatting characters.	1615
76.12	Formatting strings.	1616
76.13	String functions.	1616
76.14	Constants, types and variables	1617
76.14.1	Constants	1617
76.14.2	Types	1625
76.14.3	Variables	1642
76.15	Procedures and functions	1647
76.15.1	AbandonSignalHandler	1647
76.15.2	Abort	1647
76.15.3	AddDisk	1647
76.15.4	AddTerminateProc	1648
76.15.5	AdjustLineBreaks	1648
76.15.6	AnsiCompareFileName	1649
76.15.7	AnsiCompareStr	1649

76.15.8	AnsiCompareText	1650
76.15.9	AnsiDequotedStr	1651
76.15.10	AnsiExtractQuotedStr	1651
76.15.11	AnsiLastChar	1652
76.15.12	AnsiLowerCase	1653
76.15.13	AnsiLowerCaseFileName	1653
76.15.14	AnsiPos	1654
76.15.15	AnsiQuotedStr	1654
76.15.16	AnsiSameStr	1654
76.15.17	AnsiSameText	1654
76.15.18	AnsiStrComp	1655
76.15.19	AnsiStrIComp	1655
76.15.20	AnsiStrLastChar	1656
76.15.21	AnsiStrLComp	1657
76.15.22	AnsiStrLIComp	1658
76.15.23	AnsiStrLower	1659
76.15.24	AnsiStrPos	1659
76.15.25	AnsiStrRScan	1660
76.15.26	AnsiStrScan	1660
76.15.27	AnsiStrUpper	1660
76.15.28	AnsiUpperCase	1661
76.15.29	AnsiUpperCaseFileName	1661
76.15.30	AppendStr	1662
76.15.31	ApplicationName	1662
76.15.32	ArrayOfConstToStr	1663
76.15.33	ArrayOfConstToStrArray	1663
76.15.34	AssignStr	1663
76.15.35	BCDToInt	1663
76.15.36	Beep	1664
76.15.37	BoolToStr	1664
76.15.38	ByteLength	1665
76.15.39	BytesOf	1665
76.15.40	ByteToCharIndex	1665
76.15.41	ByteToCharLen	1665
76.15.42	ByteType	1666
76.15.43	CallTerminateProcs	1666
76.15.44	ChangeFileExt	1666
76.15.45	ChangeFilePath	1666
76.15.46	CharInSet	1667
76.15.47	CharToByteLen	1667

76.15.48	CheckOSError	1667
76.15.49	CodePageNameToCodePage	1667
76.15.50	CodePageToCodePageName	1668
76.15.51	CompareMem	1668
76.15.52	CompareMemRange	1668
76.15.53	CompareStr	1668
76.15.54	CompareText	1669
76.15.55	ComposeDateTime	1670
76.15.56	ConcatPaths	1671
76.15.57	CreateDir	1671
76.15.58	CreateGUID	1672
76.15.59	CurrentYear	1672
76.15.60	CurrToStr	1673
76.15.61	CurrToStrF	1673
76.15.62	Date	1673
76.15.63	DateTimeToFileDate	1674
76.15.64	DateTimeToStr	1674
76.15.65	DateTimeToString	1675
76.15.66	DateTimeToSystemTime	1676
76.15.67	DateTimeToTimeStamp	1677
76.15.68	DateToStr	1677
76.15.69	DayOfWeek	1678
76.15.70	DecodeDate	1678
76.15.71	DecodeDateFully	1679
76.15.72	DecodeTime	1679
76.15.73	DeleteFile	1680
76.15.74	DirectoryExists	1680
76.15.75	DiskFree	1681
76.15.76	DiskSize	1682
76.15.77	DisposeStr	1682
76.15.78	DoDirSeparators	1682
76.15.79	EncodeDate	1683
76.15.80	EncodeTime	1684
76.15.81	ExceptAddr	1684
76.15.82	ExceptFrameCount	1685
76.15.83	ExceptFrames	1685
76.15.84	ExceptionErrorMessage	1685
76.15.85	ExceptObject	1685
76.15.86	Exchange	1686
76.15.87	ExcludeLeadingPathDelimiter	1686

76.15.88 ExcludeTrailingBackslash	1686
76.15.89 ExcludeTrailingPathDelimiter	1687
76.15.90 ExecuteProcess	1687
76.15.91 ExeSearch	1687
76.15.92 ExpandFileName	1688
76.15.93 ExpandFileNameCase	1689
76.15.94 ExpandUNCFileName	1689
76.15.95 ExtractFileDir	1690
76.15.96 ExtractFileDrive	1691
76.15.97 ExtractFileExt	1691
76.15.98 ExtractFileName	1691
76.15.99 ExtractFilePath	1692
76.15.100ExtractRelativePath	1692
76.15.101ExtractShortPathName	1693
76.15.102FileAge	1693
76.15.103FileAgeUTC	1694
76.15.104FileClose	1694
76.15.105FileCreate	1694
76.15.106FileDateToDateTime	1696
76.15.107FileDateToUniversal	1697
76.15.108FileExists	1697
76.15.109FileFlush	1697
76.15.110FileGetAttr	1698
76.15.111FileGetDate	1699
76.15.112FileGetDateTimeInfo	1699
76.15.113FileGetDateUTC	1700
76.15.114FileGetSymLinkTarget	1700
76.15.115FileIsReadOnly	1700
76.15.116FileOpen	1700
76.15.117FileRead	1701
76.15.118FileSearch	1702
76.15.119FileSeek	1703
76.15.120FileSetAttr	1704
76.15.121FileSetDate	1704
76.15.122FileSetDateUTC	1705
76.15.123FileTruncate	1705
76.15.124FileWrite	1705
76.15.125FindClose	1706
76.15.126FindCmdLineSwitch	1706
76.15.127FindFirst	1706

76.15.128FindNext	1708
76.15.129FloatToCurr	1708
76.15.130FloatToDateTime	1708
76.15.131FloatToDecimal	1709
76.15.132FloatToStr	1709
76.15.133FloatToStrF	1710
76.15.134FloatToText	1713
76.15.135FloatToTextFmt	1714
76.15.136FmtStr	1714
76.15.137ForceDirectories	1715
76.15.138Format	1715
76.15.139FormatBuf	1722
76.15.140FormatCurr	1723
76.15.141FormatDateTime	1723
76.15.142FormatFloat	1724
76.15.143FreeAndNil	1725
76.15.144FreeMemAndNil	1726
76.15.145GetAppConfigDir	1726
76.15.146GetAppConfigFile	1726
76.15.147GetCompiledArchitecture	1727
76.15.148GetCompiledPlatform	1727
76.15.149GetCurrentDir	1727
76.15.150GetDirs	1728
76.15.151GetDriveIDFromLetter	1729
76.15.152GetEnvironmentString	1729
76.15.153GetEnvironmentVariable	1729
76.15.154GetEnvironmentVariableCount	1729
76.15.155GetFileAsString	1730
76.15.156GetFileContents	1730
76.15.157GetFileHandle	1731
76.15.158GetLastError	1731
76.15.159GetLocalTime	1731
76.15.160GetLocalTimeOffset	1732
76.15.161GetModuleName	1732
76.15.162GetTempDir	1732
76.15.163GetTempFileName	1733
76.15.164GetTickCount	1733
76.15.165GetTickCount64	1734
76.15.166GetUniversalTime	1734
76.15.167GetUserDir	1734

76.15.168GuidCase	1734
76.15.169GUIDToString	1734
76.15.170HashName	1735
76.15.171HookSignal	1735
76.15.172IfThen	1735
76.15.173IncAMonth	1735
76.15.174IncludeLeadingPathDelimiter	1736
76.15.175IncludeTrailingBackslash	1736
76.15.176IncludeTrailingPathDelimiter	1736
76.15.177IncMonth	1737
76.15.178InquireSignal	1737
76.15.179IntToHex	1738
76.15.180IntToStr	1738
76.15.181IsDelimiter	1739
76.15.182IsEqualGUID	1739
76.15.183IsFileNameCasePreserving	1740
76.15.184IsFileNameCaseSensitive	1740
76.15.185IsLeadChar	1740
76.15.186IsLeapYear	1740
76.15.187IsPathDelimiter	1741
76.15.188IsValidIdent	1741
76.15.189LastDelimiter	1742
76.15.190LeftStr	1742
76.15.191ListIndexError	1743
76.15.192ListIndexErrorMsg	1743
76.15.193LoadStr	1743
76.15.194LocalTimeToUniversal	1743
76.15.195LowerCase	1743
76.15.196MSecsToTimeStamp	1744
76.15.197NewStr	1745
76.15.198Now	1745
76.15.199NowUTC	1746
76.15.200OutOfMemoryError	1746
76.15.201QuotedStr	1746
76.15.202RaiseLastError	1747
76.15.203RangeIndexError	1747
76.15.204RemoveDir	1747
76.15.205RenameFile	1747
76.15.206ReplaceDate	1748
76.15.207ReplaceTime	1748

76.15.208RightStr	1748
76.15.209SafeFormat	1749
76.15.210SafeLoadLibrary	1749
76.15.211SameFileName	1749
76.15.212SameStr	1750
76.15.213SameText	1750
76.15.214SetCurrentDir	1750
76.15.215SetDirSeparators	1751
76.15.216ShowException	1751
76.15.217Sleep	1751
76.15.218SScanf	1752
76.15.219StrAlloc	1753
76.15.220StrBufSize	1753
76.15.221StrByteType	1754
76.15.222strcat	1754
76.15.223StrCharLength	1754
76.15.224strcomp	1755
76.15.225StrCopy	1755
76.15.226StrDispose	1756
76.15.227strecopy	1756
76.15.228strend	1757
76.15.229StrFmt	1757
76.15.230stricomp	1758
76.15.231StringOf	1758
76.15.232StringReplace	1759
76.15.233StringToGUID	1759
76.15.234strlcat	1760
76.15.235strlcomp	1760
76.15.236StrLCopy	1761
76.15.237StrLen	1762
76.15.238StrLFmt	1762
76.15.239strlicomp	1763
76.15.240strlower	1763
76.15.241StrMove	1764
76.15.242strnew	1764
76.15.243StrNextChar	1765
76.15.244StrPas	1765
76.15.245StrPCopy	1766
76.15.246StrPLCopy	1766
76.15.247strpos	1766

76.15.248strrscan	1767
76.15.249StrScan	1767
76.15.250StrToBool	1768
76.15.251StrToBoolDef	1768
76.15.252StrToCurr	1768
76.15.253StrToCurrDef	1769
76.15.254StrToDate	1769
76.15.255StrToDateDef	1770
76.15.256StrToDateTime	1770
76.15.257StrToDateTimeDef	1771
76.15.258StrToDWord	1771
76.15.259StrToDWordDef	1772
76.15.260StrToFloat	1772
76.15.261StrToFloatDef	1773
76.15.262StrToInt	1773
76.15.263StrToInt64	1774
76.15.264StrToInt64Def	1775
76.15.265StrToIntDef	1775
76.15.266StrToQWord	1776
76.15.267StrToQWordDef	1776
76.15.268StrToTime	1776
76.15.269StrToTimeDef	1777
76.15.270StrToUInt	1777
76.15.271StrToUInt64	1778
76.15.272StrToUInt64Def	1778
76.15.273StrToUIntDef	1778
76.15.274strupper	1779
76.15.275Supports	1779
76.15.276SysErrorMessage	1779
76.15.277SystemTimeToDateTime	1780
76.15.278TextToFloat	1780
76.15.279Time	1781
76.15.280TimeStampToDateTime	1782
76.15.281TimeStampToMsecs	1783
76.15.282TimeToStr	1783
76.15.283Trim	1783
76.15.284TrimLeft	1784
76.15.285TrimRight	1785
76.15.286TryEncodeDate	1785
76.15.287TryEncodeTime	1786

76.15.288TryFloatToCurr	1786
76.15.289TryStringToGUID	1786
76.15.290TryStrToBool	1787
76.15.291TryStrToCurr	1787
76.15.292TryStrToDate	1787
76.15.293TryStrToDateTime	1788
76.15.294TryStrToDWord	1788
76.15.295TryStrToFloat	1789
76.15.296TryStrToInt	1789
76.15.297TryStrToInt64	1789
76.15.298TryStrToQWord	1790
76.15.299TryStrToTime	1790
76.15.300TryStrToUInt	1790
76.15.301TryStrToUInt64	1791
76.15.302UIntToStr	1791
76.15.303UnhookSignal	1791
76.15.304UnicodeCompareStr	1792
76.15.305UnicodeCompareText	1792
76.15.306UnicodeFmtStr	1792
76.15.307UnicodeFormat	1793
76.15.308UnicodeFormatBuf	1793
76.15.309UnicodeLowerCase	1793
76.15.310UnicodeSameStr	1794
76.15.311UnicodeSameText	1794
76.15.312UnicodeStringReplace	1794
76.15.313UnicodeUpperCase	1795
76.15.314UniversalTimeToLocal	1795
76.15.315UniversalToFileDate	1795
76.15.316UpperCase	1795
76.15.317VendorName	1796
76.15.318WideBytesOf	1796
76.15.319WideCompareStr	1796
76.15.320WideCompareText	1797
76.15.321WideFmtStr	1797
76.15.322WideFormat	1797
76.15.323WideFormatBuf	1798
76.15.324WideLowerCase	1798
76.15.325WideSameStr	1798
76.15.326WideSameText	1798
76.15.327WideStrAlloc	1799

76.15.328	WideStringOf	1799
76.15.329	WideStringReplace	1799
76.15.330	WideUpperCase	1800
76.15.331	WrapText	1800
76.16	TDateTimeInfoRec	1800
76.16.1	Property overview	1801
76.16.2	TDateTimeInfoRec.CreationTime	1801
76.16.3	TDateTimeInfoRec.LastAccessTime	1801
76.16.4	TDateTimeInfoRec.TimeStamp	1801
76.17	TFormatSettings	1801
76.17.1	Method overview	1802
76.17.2	TFormatSettings.Invariant	1802
76.17.3	TFormatSettings.Create	1802
76.18	TOSVersion	1802
76.18.1	Method overview	1804
76.18.2	Property overview	1804
76.18.3	TOSVersion.Check	1804
76.18.4	TOSVersion.ToString	1804
76.18.5	TOSVersion.Architecture	1805
76.18.6	TOSVersion.Build	1805
76.18.7	TOSVersion.Major	1805
76.18.8	TOSVersion.Minor	1805
76.18.9	TOSVersion.Name	1805
76.18.10	TOSVersion.Platform	1805
76.18.11	TOSVersion.ServicePackMajor	1805
76.18.12	TOSVersion.ServicePackMinor	1806
76.18.13	TOSVersion.PrettyName	1806
76.18.14	TOSVersion.LibCVersionMajor	1806
76.18.15	TOSVersion.LibCVersionMinor	1806
76.19	TRawbyteSearchRec	1806
76.19.1	Method overview	1807
76.19.2	Property overview	1807
76.19.3	TRawbyteSearchRec.IsDirectory	1807
76.19.4	TRawbyteSearchRec.IsCurrentOrParentDir	1807
76.19.5	TRawbyteSearchRec.TimeStamp	1807
76.19.6	TRawbyteSearchRec.TimeStampUTC	1808
76.20	TRawbyteSymLinkRec	1808
76.20.1	Property overview	1808
76.20.2	TRawbyteSymLinkRec.TimeStamp	1808
76.21	TUnicodeSearchRec	1809

76.21.1	Method overview	1809
76.21.2	Property overview	1809
76.21.3	TUnicodeSearchRec.IsDirectory	1809
76.21.4	TUnicodeSearchRec.IsCurrentOrParentDir	1810
76.21.5	TUnicodeSearchRec.TimeStamp	1810
76.21.6	TUnicodeSearchRec.TimeStampUTC	1810
76.22	TUnicodeSymLinkRec	1810
76.22.1	Property overview	1811
76.22.2	TUnicodeSymLinkRec.TimeStamp	1811
76.23	EAbort	1811
76.23.1	Description	1811
76.24	EAbstractError	1811
76.24.1	Description	1811
76.25	EAccessViolation	1811
76.25.1	Description	1811
76.26	EArgumentException	1811
76.26.1	Description	1811
76.27	EArgumentNilException	1812
76.27.1	Description	1812
76.28	EArgumentOutOfRangeException	1812
76.28.1	Description	1812
76.29	EAssertionFailed	1812
76.29.1	Description	1812
76.30	EBusError	1812
76.30.1	Description	1812
76.31	ECFError	1812
76.32	EControlC	1812
76.32.1	Description	1812
76.33	EConvertError	1812
76.33.1	Description	1812
76.34	EDirectoryNotFoundException	1813
76.34.1	Description	1813
76.35	EDivByZero	1813
76.35.1	Description	1813
76.36	EEncodingError	1813
76.36.1	Description	1813
76.37	EExternal	1813
76.37.1	Description	1813
76.38	EExternalException	1813
76.38.1	Description	1813

76.39	EFileNotFoundException	1813
76.39.1	Description	1813
76.40	EFormatError	1814
76.40.1	Description	1814
76.41	EHeapMemoryError	1814
76.41.1	Description	1814
76.41.2	Method overview	1814
76.41.3	EHeapMemoryError.FreeInstance	1814
76.42	EInOutArgumentException	1814
76.42.1	Method overview	1814
76.42.2	EInOutArgumentException.Create	1814
76.42.3	EInOutArgumentException.CreateRes	1815
76.42.4	EInOutArgumentException.CreateFmt	1815
76.43	EInOutError	1815
76.43.1	Description	1815
76.44	EIntError	1815
76.44.1	Description	1815
76.45	EIntfCastError	1815
76.45.1	Description	1815
76.46	EIntOverflow	1815
76.46.1	Description	1815
76.47	EInvalidCast	1816
76.47.1	Description	1816
76.48	EInvalidContainer	1816
76.48.1	Description	1816
76.49	EInvalidInsert	1816
76.49.1	Description	1816
76.50	EInvalidOp	1816
76.50.1	Description	1816
76.51	EInvalidOpException	1816
76.51.1	Description	1816
76.52	EInvalidPointer	1816
76.52.1	Description	1816
76.53	EListError	1817
76.54	EMathError	1817
76.54.1	Description	1817
76.55	EMonitor	1817
76.56	EMonitorLockException	1817
76.57	ENoConstructException	1817
76.57.1	Description	1817

76.58	ENoDynLibsSupport	1817
76.58.1	Description	1817
76.59	ENoMonitorSupportException	1817
76.60	ENoThreadSupport	1817
76.60.1	Description	1817
76.61	ENotImplemented	1817
76.61.1	Description	1817
76.62	ENotSupportedException	1818
76.62.1	Description	1818
76.63	ENoWideStringSupport	1818
76.63.1	Description	1818
76.64	EObjectCheck	1818
76.64.1	Description	1818
76.65	EObjectDisposed	1818
76.66	EOperationCancelled	1818
76.67	EOSError	1818
76.67.1	Description	1818
76.68	EOutOfMemory	1818
76.68.1	Description	1818
76.69	EOverflow	1819
76.69.1	Description	1819
76.70	EPackageError	1819
76.70.1	Description	1819
76.71	EPathNotFoundException	1819
76.71.1	Description	1819
76.72	EPathTooLongException	1819
76.72.1	Description	1819
76.73	EPrivilege	1819
76.73.1	Description	1819
76.74	EProgrammerNotFound	1819
76.74.1	Description	1819
76.75	EPropReadOnly	1820
76.75.1	Description	1820
76.76	EPropWriteOnly	1820
76.76.1	Description	1820
76.77	ERangeError	1820
76.77.1	Description	1820
76.78	ESafecallException	1820
76.78.1	Description	1820
76.79	ESigQuit	1820

76.79.1	Description	1820
76.80	EStackOverflow	1820
76.80.1	Description	1820
76.81	EThreadError	1821
76.82	EUnderflow	1821
76.82.1	Description	1821
76.83	EVariantError	1821
76.83.1	Description	1821
76.83.2	Method overview	1821
76.83.3	EVariantError.CreateCode	1821
76.84	Exception	1821
76.84.1	Description	1821
76.84.2	Method overview	1822
76.84.3	Property overview	1822
76.84.4	Exception.Create	1822
76.84.5	Exception.CreateFmt	1822
76.84.6	Exception.CreateRes	1822
76.84.7	Exception.CreateResFmt	1823
76.84.8	Exception.CreateHelp	1823
76.84.9	Exception.CreateFmtHelp	1823
76.84.10	Exception.CreateResHelp	1823
76.84.11	Exception.CreateResFmtHelp	1824
76.84.12	Exception.ToString	1824
76.84.13	Exception.GetBaseException	1824
76.84.14	Exception.HelpContext	1824
76.84.15	Exception.Message	1824
76.85	EZeroDivide	1825
76.85.1	Description	1825
76.86	IReadWriteSync	1825
76.86.1	Description	1825
76.86.2	Method overview	1825
76.86.3	IReadWriteSync.BeginRead	1825
76.86.4	IReadWriteSync.EndRead	1825
76.86.5	IReadWriteSync.BeginWrite	1826
76.86.6	IReadWriteSync.EndWrite	1826
76.87	TANSISTRINGBUILDER	1826
76.87.1	Description	1826
76.87.2	Method overview	1827
76.87.3	Property overview	1827
76.87.4	TANSISTRINGBUILDER.Create	1827

76.87.5	TANSISTRINGBUILDER.Append	1827
76.87.6	TANSISTRINGBUILDER.AppendFormat	1828
76.87.7	TANSISTRINGBUILDER.AppendLine	1829
76.87.8	TANSISTRINGBUILDER.Clear	1829
76.87.9	TANSISTRINGBUILDER.CopyTo	1829
76.87.10	TANSISTRINGBUILDER.EnsureCapacity	1829
76.87.11	TANSISTRINGBUILDER.Equals	1830
76.87.12	TANSISTRINGBUILDER.GetEnumerator	1830
76.87.13	TANSISTRINGBUILDER.Insert	1830
76.87.14	TANSISTRINGBUILDER.Remove	1831
76.87.15	TANSISTRINGBUILDER.Replace	1831
76.87.16	TANSISTRINGBUILDER.ToString	1832
76.87.17	TANSISTRINGBUILDER.Chars	1832
76.87.18	TANSISTRINGBUILDER.Length	1832
76.87.19	TANSISTRINGBUILDER.Capacity	1833
76.87.20	TANSISTRINGBUILDER.MaxCapacity	1833
76.88	TAnsiStringHelper	1835
76.88.1	Method overview	1835
76.88.2	Property overview	1836
76.88.3	TAnsiStringHelper.Compare	1836
76.88.4	TAnsiStringHelper.CompareOrdinal	1836
76.88.5	TAnsiStringHelper.CompareText	1836
76.88.6	TAnsiStringHelper.Copy	1836
76.88.7	TAnsiStringHelper.Create	1837
76.88.8	TAnsiStringHelper.EndsWith	1837
76.88.9	TAnsiStringHelper.Equals	1837
76.88.10	TAnsiStringHelper.Format	1837
76.88.11	TAnsiStringHelper.IsNullOrEmpty	1837
76.88.12	TAnsiStringHelper.IsNullOrEmptyOrWhiteSpace	1837
76.88.13	TAnsiStringHelper.Join	1838
76.88.14	TAnsiStringHelper.LowerCase	1838
76.88.15	TAnsiStringHelper.Parse	1838
76.88.16	TAnsiStringHelper.ToBoolean	1838
76.88.17	TAnsiStringHelper.ToDouble	1838
76.88.18	TAnsiStringHelper.ToExtended	1839
76.88.19	TAnsiStringHelper.ToInt64	1839
76.88.20	TAnsiStringHelper.ToInteger	1839
76.88.21	TAnsiStringHelper.ToSingle	1839
76.88.22	TAnsiStringHelper.UpperCase	1839
76.88.23	TAnsiStringHelper.CompareTo	1839

76.88.24	TAnsiStringHelper.Contains	1839
76.88.25	TAnsiStringHelper.CopyTo	1840
76.88.26	TAnsiStringHelper.CountChar	1840
76.88.27	TAnsiStringHelper.DeQuotedString	1840
76.88.28	TAnsiStringHelper.EndsWith	1840
76.88.29	TAnsiStringHelper.GetHashCode	1840
76.88.30	TAnsiStringHelper.IndexOf	1840
76.88.31	TAnsiStringHelper.IndexOfUnQuoted	1841
76.88.32	TAnsiStringHelper.IndexOfAny	1841
76.88.33	TAnsiStringHelper.IndexOfAnyUnquoted	1841
76.88.34	TAnsiStringHelper.Insert	1841
76.88.35	TAnsiStringHelper.IsDelimiter	1842
76.88.36	TAnsiStringHelper.IsEmpty	1842
76.88.37	TAnsiStringHelper.LastDelimiter	1842
76.88.38	TAnsiStringHelper.LastIndexOf	1842
76.88.39	TAnsiStringHelper.LastIndexOfAny	1842
76.88.40	TAnsiStringHelper.PadLeft	1842
76.88.41	TAnsiStringHelper.PadRight	1843
76.88.42	TAnsiStringHelper.QuotedString	1843
76.88.43	TAnsiStringHelper.Remove	1843
76.88.44	TAnsiStringHelper.Replace	1843
76.88.45	TAnsiStringHelper.Split	1843
76.88.46	TAnsiStringHelper.StartsWith	1844
76.88.47	TAnsiStringHelper.Substring	1844
76.88.48	TAnsiStringHelper.ToCharArray	1845
76.88.49	TAnsiStringHelper.ToLower	1845
76.88.50	TAnsiStringHelper.ToLowerInvariant	1845
76.88.51	TAnsiStringHelper.ToUpper	1845
76.88.52	TAnsiStringHelper.ToUpperInvariant	1845
76.88.53	TAnsiStringHelper.Trim	1845
76.88.54	TAnsiStringHelper.TrimLeft	1845
76.88.55	TAnsiStringHelper.TrimRight	1846
76.88.56	TAnsiStringHelper.TrimEnd	1846
76.88.57	TAnsiStringHelper.TrimStart	1846
76.88.58	TAnsiStringHelper.Chars	1846
76.88.59	TAnsiStringHelper.Length	1846
76.89	TBigEndianUnicodeEncoding	1846
76.89.1	Description	1846
76.89.2	Method overview	1846
76.89.3	TBigEndianUnicodeEncoding.Clone	1847

76.89.4	TBigEndianUnicodeEncoding.GetPreamble	1847
76.90	TBooleanHelper	1847
76.90.1	Description	1847
76.90.2	Method overview	1847
76.90.3	TBooleanHelper.Parse	1847
76.90.4	TBooleanHelper.Size	1848
76.90.5	TBooleanHelper.ToString	1848
76.90.6	TBooleanHelper.TryParse	1848
76.90.7	TBooleanHelper.ToInteger	1849
76.91	TByteBoolHelper	1849
76.91.1	Description	1849
76.91.2	Method overview	1849
76.91.3	TByteBoolHelper.Parse	1849
76.91.4	TByteBoolHelper.Size	1849
76.91.5	TByteBoolHelper.ToString	1850
76.91.6	TByteBoolHelper.TryParse	1850
76.91.7	TByteBoolHelper.ToInteger	1850
76.92	TByteHelper	1851
76.92.1	Description	1851
76.92.2	Method overview	1851
76.92.3	Property overview	1851
76.92.4	TByteHelper.Parse	1851
76.92.5	TByteHelper.Size	1852
76.92.6	TByteHelper.ToString	1852
76.92.7	TByteHelper.TryParse	1852
76.92.8	TByteHelper.ToBoolean	1852
76.92.9	TByteHelper.ToDouble	1853
76.92.10	TByteHelper.ToExtended	1853
76.92.11	TByteHelper.ToBinString	1853
76.92.12	TByteHelper.ToHexString	1853
76.92.13	TByteHelper.ToSingle	1854
76.92.14	TByteHelper.SetBit	1854
76.92.15	TByteHelper.ClearBit	1854
76.92.16	TByteHelper.ToggleBit	1854
76.92.17	TByteHelper.TestBit	1855
76.92.18	TByteHelper.Clear	1855
76.92.19	TByteHelper.HighestSetBitPos	1855
76.92.20	TByteHelper.LowestSetBitPos	1855
76.92.21	TByteHelper.SetBitsCount	1856
76.92.22	TByteHelper.Bits	1856

76.92.23	TByteHelper.Nibbles	1856
76.93	TCardinalHelper	1857
76.93.1	Description	1857
76.93.2	Method overview	1857
76.93.3	Property overview	1857
76.93.4	TCardinalHelper.Parse	1857
76.93.5	TCardinalHelper.Size	1858
76.93.6	TCardinalHelper.ToString	1858
76.93.7	TCardinalHelper.TryParse	1858
76.93.8	TCardinalHelper.ToBoolean	1858
76.93.9	TCardinalHelper.ToDouble	1859
76.93.10	TCardinalHelper.ToExtended	1859
76.93.11	TCardinalHelper.ToBinString	1859
76.93.12	TCardinalHelper.ToHexString	1859
76.93.13	TCardinalHelper.ToSingle	1860
76.93.14	TCardinalHelper.SetBit	1860
76.93.15	TCardinalHelper.ClearBit	1860
76.93.16	TCardinalHelper.ToggleBit	1860
76.93.17	TCardinalHelper.TestBit	1861
76.93.18	TCardinalHelper.Clear	1861
76.93.19	TCardinalHelper.HighestSetBitPos	1861
76.93.20	TCardinalHelper.LowestSetBitPos	1861
76.93.21	TCardinalHelper.SetBitsCount	1862
76.93.22	TCardinalHelper.Bits	1862
76.93.23	TCardinalHelper.Nibbles	1862
76.93.24	TCardinalHelper.Bytes	1863
76.93.25	TCardinalHelper.Words	1863
76.94	TCurrencyHelper	1863
76.94.1	Method overview	1863
76.94.2	Property overview	1863
76.94.3	TCurrencyHelper.Ceil	1864
76.94.4	TCurrencyHelper.Floor	1864
76.94.5	TCurrencyHelper.Frac	1864
76.94.6	TCurrencyHelper.Parse	1864
76.94.7	TCurrencyHelper.Size	1864
76.94.8	TCurrencyHelper.ToString	1864
76.94.9	TCurrencyHelper.Trunc	1864
76.94.10	TCurrencyHelper.TryParse	1865
76.94.11	TCurrencyHelper.MaxValue	1865
76.94.12	TCurrencyHelper.MinValue	1865

76.95	TDoubleHelper	1865
76.95.1	Description	1865
76.95.2	Method overview	1865
76.95.3	Property overview	1866
76.95.4	TDoubleHelper.IsInfinity	1866
76.95.5	TDoubleHelper.IsNan	1866
76.95.6	TDoubleHelper.IsNegativeInfinity	1866
76.95.7	TDoubleHelper.IsPositiveInfinity	1867
76.95.8	TDoubleHelper.Parse	1867
76.95.9	TDoubleHelper.Size	1867
76.95.10	TDoubleHelper.ToString	1867
76.95.11	TDoubleHelper.TryParse	1868
76.95.12	TDoubleHelper.BuildUp	1868
76.95.13	TDoubleHelper.Exponent	1869
76.95.14	TDoubleHelper.Fraction	1869
76.95.15	TDoubleHelper.Mantissa	1869
76.95.16	TDoubleHelper.SpecialType	1870
76.95.17	TDoubleHelper.Bytes	1870
76.95.18	TDoubleHelper.Words	1870
76.95.19	TDoubleHelper.Sign	1870
76.95.20	TDoubleHelper.Exp	1871
76.95.21	TDoubleHelper.Frac	1871
76.96	TEncoding	1871
76.96.1	Description	1871
76.96.2	Method overview	1872
76.96.3	Property overview	1872
76.96.4	TEncoding.Clone	1872
76.96.5	TEncoding.Convert	1872
76.96.6	TEncoding.IsStandardEncoding	1873
76.96.7	TEncoding.GetBufferEncoding	1873
76.96.8	TEncoding.GetEncoding	1873
76.96.9	TEncoding.GetMaxByteCount	1874
76.96.10	TEncoding.GetMaxCharCount	1874
76.96.11	TEncoding.GetPreamble	1874
76.96.12	TEncoding.GetString	1875
76.96.13	TEncoding.CodePage	1875
76.96.14	TEncoding.EncodingName	1875
76.96.15	TEncoding.IsSingleByte	1875
76.96.16	TEncoding.ANSI	1876
76.96.17	TEncoding.ASCII	1876

76.96.18	TEncoding.BigEndianUnicode	1876
76.96.19	TEncoding.Default	1876
76.96.20	TEncoding.SystemEncoding	1877
76.96.21	TEncoding.Unicode	1877
76.96.22	TEncoding.UTF7	1877
76.96.23	TEncoding.UTF8	1878
76.97	TExtendedHelper	1878
76.97.1	Description	1878
76.97.2	Method overview	1878
76.97.3	Property overview	1878
76.97.4	TExtendedHelper.ToString	1879
76.97.5	TExtendedHelper.Parse	1879
76.97.6	TExtendedHelper.TryParse	1880
76.97.7	TExtendedHelper.IsNan	1880
76.97.8	TExtendedHelper.IsInfinity	1880
76.97.9	TExtendedHelper.IsNegativeInfinity	1881
76.97.10	TExtendedHelper.IsPositiveInfinity	1881
76.97.11	TExtendedHelper.Size	1881
76.97.12	TExtendedHelper.BuildUp	1881
76.97.13	TExtendedHelper.Exponent	1882
76.97.14	TExtendedHelper.Fraction	1882
76.97.15	TExtendedHelper.Mantissa	1882
76.97.16	TExtendedHelper.SpecialType	1882
76.97.17	TExtendedHelper.Bytes	1883
76.97.18	TExtendedHelper.Words	1883
76.97.19	TExtendedHelper.Sign	1883
76.97.20	TExtendedHelper.Exp	1883
76.97.21	TExtendedHelper.Frac	1884
76.98	TGuidHelper	1884
76.98.1	Description	1884
76.98.2	Method overview	1884
76.98.3	TGuidHelper.Create	1884
76.98.4	TGuidHelper.NewGuid	1885
76.98.5	TGuidHelper.ToByteArray	1885
76.98.6	TGuidHelper.ToString	1885
76.99	TInt64Helper	1886
76.99.1	Description	1886
76.99.2	Method overview	1886
76.99.3	Property overview	1886
76.99.4	TInt64Helper.Parse	1886

76.99.5	TInt64Helper.Size	1887
76.99.6	TInt64Helper.ToString	1887
76.99.7	TInt64Helper.TryParse	1887
76.99.8	TInt64Helper.ToBoolean	1887
76.99.9	TInt64Helper.ToDouble	1888
76.99.10	TInt64Helper.ToExtended	1888
76.99.11	TInt64Helper.ToBinString	1888
76.99.12	TInt64Helper.ToHexString	1888
76.99.13	TInt64Helper.ToSingle	1889
76.99.14	TInt64Helper.SetBit	1889
76.99.15	TInt64Helper.ClearBit	1889
76.99.16	TInt64Helper.ToggleBit	1889
76.99.17	TInt64Helper.TestBit	1890
76.99.18	TInt64Helper.Clear	1890
76.99.19	TInt64Helper.HighestSetBitPos	1890
76.99.20	TInt64Helper.LowestSetBitPos	1890
76.99.21	TInt64Helper.SetBitsCount	1891
76.99.22	TInt64Helper.Bits	1891
76.99.23	TInt64Helper.Nibbles	1891
76.99.24	TInt64Helper.Bytes	1892
76.99.25	TInt64Helper.Words	1892
76.99.26	TInt64Helper.DWords	1892
76.100	TIntegerHelper	1893
76.100.1	Description	1893
76.100.2	Method overview	1893
76.100.3	Property overview	1893
76.100.4	TIntegerHelper.Size	1893
76.100.5	TIntegerHelper.ToString	1894
76.100.6	TIntegerHelper.Parse	1894
76.100.7	TIntegerHelper.TryParse	1894
76.100.8	TIntegerHelper.ToBoolean	1894
76.100.9	TIntegerHelper.ToDouble	1895
76.100.10	TIntegerHelper.ToExtended	1895
76.100.11	TIntegerHelper.ToBinString	1895
76.100.12	TIntegerHelper.ToHexString	1895
76.100.13	TIntegerHelper.ToSingle	1896
76.100.14	TIntegerHelper.SetBit	1896
76.100.15	TIntegerHelper.ClearBit	1896
76.100.16	TIntegerHelper.ToggleBit	1896
76.100.17	TIntegerHelper.TestBit	1897

76.100.18TIntegerHelper.Clear	1897
76.100.19TIntegerHelper.HighestSetBitPos	1897
76.100.20TIntegerHelper.LowestSetBitPos	1897
76.100.21TIntegerHelper.SetBitsCount	1898
76.100.22TIntegerHelper.Bits	1898
76.100.23TIntegerHelper.Nibbles	1898
76.100.24TIntegerHelper.Bytes	1899
76.101TLongBoolHelper	1899
76.101.1 Description	1899
76.101.2 Method overview	1899
76.101.3 TLongBoolHelper.Parse	1899
76.101.4 TLongBoolHelper.Size	1900
76.101.5 TLongBoolHelper.ToString	1900
76.101.6 TLongBoolHelper.TryToParse	1900
76.101.7 TLongBoolHelper.ToInteger	1900
76.102TLongIntHelper	1901
76.102.1 Method overview	1901
76.102.2 Property overview	1901
76.102.3 TLongIntHelper.Size	1901
76.102.4 TLongIntHelper.ToString	1901
76.102.5 TLongIntHelper.Parse	1901
76.102.6 TLongIntHelper.TryParse	1902
76.102.7 TLongIntHelper.ToBoolean	1902
76.102.8 TLongIntHelper.ToDouble	1902
76.102.9 TLongIntHelper.ToExtended	1902
76.102.10TLongIntHelper.ToBinString	1902
76.102.11TLongIntHelper.ToHexString	1902
76.102.12TLongIntHelper.ToSingle	1902
76.102.13TLongIntHelper.SetBit	1902
76.102.14TLongIntHelper.ClearBit	1903
76.102.15TLongIntHelper.ToggleBit	1903
76.102.16TLongIntHelper.TestBit	1903
76.102.17TLongIntHelper.Clear	1903
76.102.18TLongIntHelper.HighestSetBitPos	1903
76.102.19TLongIntHelper.LowestSetBitPos	1903
76.102.20TLongIntHelper.SetBitsCount	1903
76.102.21TLongIntHelper.Bits	1903
76.102.22TLongIntHelper.Nibbles	1904
76.102.23TLongIntHelper.Bytes	1904
76.102.24TLongIntHelper.Words	1904

76.103	TMBCSEncoding	1904
76.103.1	Description	1904
76.103.2	Method overview	1904
76.103.3	TMBCSEncoding.Create	1904
76.103.4	TMBCSEncoding.Clone	1905
76.103.5	TMBCSEncoding.GetMaxByteCount	1905
76.103.6	TMBCSEncoding.GetMaxCharCount	1905
76.103.7	TMBCSEncoding.GetPreamble	1905
76.104	TMREWException	1906
76.104.1	Description	1906
76.105	TMultiReadExclusiveWriteSynchronizer	1906
76.105.1	Description	1906
76.105.2	Interfaces overview	1906
76.105.3	Method overview	1906
76.105.4	Property overview	1906
76.105.5	TMultiReadExclusiveWriteSynchronizer.Create	1906
76.105.6	TMultiReadExclusiveWriteSynchronizer.Destroy	1907
76.105.7	TMultiReadExclusiveWriteSynchronizer.Beginwrite	1907
76.105.8	TMultiReadExclusiveWriteSynchronizer.Endwrite	1907
76.105.9	TMultiReadExclusiveWriteSynchronizer.Beginread	1907
76.105.10	TMultiReadExclusiveWriteSynchronizer.Endread	1908
76.105.11	TMultiReadExclusiveWriteSynchronizer.RevisionLevel	1908
76.105.12	TMultiReadExclusiveWriteSynchronizer.WriterThreadID	1908
76.106	TNativeIntHelper	1908
76.106.1	Description	1908
76.106.2	Method overview	1909
76.106.3	Property overview	1909
76.106.4	TNativeIntHelper.Parse	1909
76.106.5	TNativeIntHelper.Size	1909
76.106.6	TNativeIntHelper.ToString	1910
76.106.7	TNativeIntHelper.TryParse	1910
76.106.8	TNativeIntHelper.ToBoolean	1910
76.106.9	TNativeIntHelper.ToDouble	1910
76.106.10	TNativeIntHelper.ToExtended	1911
76.106.11	TNativeIntHelper.ToBinString	1911
76.106.12	TNativeIntHelper.ToHexString	1911
76.106.13	TNativeIntHelper.ToSingle	1911
76.106.14	TNativeIntHelper.SetBit	1912
76.106.15	TNativeIntHelper.ClearBit	1912
76.106.16	TNativeIntHelper.ToggleBit	1912

76.106.17TNativeIntHelper.TestBit	1912
76.106.18TNativeIntHelper.Clear	1913
76.106.19TNativeIntHelper.HighestSetBitPos	1913
76.106.20TNativeIntHelper.LowestSetBitPos	1913
76.106.21TNativeIntHelper.SetBitsCount	1913
76.106.22TNativeIntHelper.Bits	1914
76.106.23TNativeIntHelper.Nibbles	1914
76.106.24TNativeIntHelper.Bytes	1914
76.107TNativeUIntHelper	1915
76.107.1 Description	1915
76.107.2 Method overview	1915
76.107.3 Property overview	1915
76.107.4 TNativeUIntHelper.Parse	1915
76.107.5 TNativeUIntHelper.Size	1916
76.107.6 TNativeUIntHelper.ToString	1916
76.107.7 TNativeUIntHelper.TryParse	1916
76.107.8 TNativeUIntHelper.ToBoolean	1916
76.107.9 TNativeUIntHelper.ToDouble	1917
76.107.10TNativeUIntHelper.ToExtended	1917
76.107.11TNativeUIntHelper.ToBinString	1917
76.107.12TNativeUIntHelper.ToHexString	1917
76.107.13TNativeUIntHelper.ToSingle	1918
76.107.14TNativeUIntHelper.SetBit	1918
76.107.15TNativeUIntHelper.ClearBit	1918
76.107.16TNativeUIntHelper.ToggleBit	1918
76.107.17TNativeUIntHelper.TestBit	1919
76.107.18TNativeUIntHelper.Clear	1919
76.107.19TNativeUIntHelper.HighestSetBitPos	1919
76.107.20TNativeUIntHelper.LowestSetBitPos	1919
76.107.21TNativeUIntHelper.SetBitsCount	1920
76.107.22TNativeUIntHelper.Bits	1920
76.107.23TNativeUIntHelper.Nibbles	1920
76.107.24TNativeUIntHelper.Bytes	1921
76.108TQWordHelper	1921
76.108.1 Description	1921
76.108.2 Method overview	1921
76.108.3 Property overview	1922
76.108.4 TQWordHelper.Parse	1922
76.108.5 TQWordHelper.Size	1922
76.108.6 TQWordHelper.ToString	1922

76.108.7 TQWordHelper.TryParse	1923
76.108.8 TQWordHelper.ToBoolean	1923
76.108.9 TQWordHelper.ToDouble	1923
76.108.10TQWordHelper.ToExtended	1923
76.108.11TQWordHelper.ToBinString	1924
76.108.12TQWordHelper.ToHexString	1924
76.108.13TQWordHelper.ToSingle	1924
76.108.14TQWordHelper.SetBit	1924
76.108.15TQWordHelper.ClearBit	1925
76.108.16TQWordHelper.ToggleBit	1925
76.108.17TQWordHelper.TestBit	1925
76.108.18TQWordHelper.Clear	1925
76.108.19TQWordHelper.HighestSetBitPos	1926
76.108.20TQWordHelper.LowestSetBitPos	1926
76.108.21TQWordHelper.SetBitsCount	1926
76.108.22TQWordHelper.Bits	1926
76.108.23TQWordHelper.Nibbles	1927
76.108.24TQWordHelper.Bytes	1927
76.108.25TQWordHelper.Words	1927
76.108.26TQWordHelper.DWords	1928
76.109TShortIntHelper	1928
76.109.1 Description	1928
76.109.2 Method overview	1929
76.109.3 Property overview	1929
76.109.4 TShortIntHelper.Parse	1929
76.109.5 TShortIntHelper.Size	1929
76.109.6 TShortIntHelper.ToString	1930
76.109.7 TShortIntHelper.TryParse	1930
76.109.8 TShortIntHelper.ToBoolean	1930
76.109.9 TShortIntHelper.ToDouble	1930
76.109.10TShortIntHelper.ToExtended	1931
76.109.11TShortIntHelper.ToBinString	1931
76.109.12TShortIntHelper.ToHexString	1931
76.109.13TShortIntHelper.ToSingle	1931
76.109.14TShortIntHelper.SetBit	1932
76.109.15TShortIntHelper.ClearBit	1932
76.109.16TShortIntHelper.ToggleBit	1932
76.109.17TShortIntHelper.TestBit	1932
76.109.18TShortIntHelper.Clear	1933
76.109.19TShortIntHelper.HighestSetBitPos	1933

76.109.20TShortIntHelper.LowestSetBitPos	1933
76.109.21TShortIntHelper.SetBitsCount	1933
76.109.22TShortIntHelper.Bits	1934
76.109.23TShortIntHelper.Nibbles	1934
76.110TShortStringHelper	1936
76.110.1 Method overview	1936
76.110.2 Property overview	1937
76.110.3 TShortStringHelper.Compare	1937
76.110.4 TShortStringHelper.CompareOrdinal	1937
76.110.5 TShortStringHelper.CompareText	1937
76.110.6 TShortStringHelper.Copy	1937
76.110.7 TShortStringHelper.Create	1938
76.110.8 TShortStringHelper.EndsText	1938
76.110.9 TShortStringHelper.Equals	1938
76.110.10TShortStringHelper.Format	1938
76.110.11TShortStringHelper.IsNullOrEmpty	1938
76.110.12TShortStringHelper.IsNullOrEmptyOrWhiteSpace	1938
76.110.13TShortStringHelper.Join	1939
76.110.14TShortStringHelper.LowerCase	1939
76.110.15TShortStringHelper.Parse	1939
76.110.16TShortStringHelper.ToBoolean	1939
76.110.17TShortStringHelper.ToDouble	1939
76.110.18TShortStringHelper.ToExtended	1940
76.110.19TShortStringHelper.ToInt64	1940
76.110.20TShortStringHelper.ToInteger	1940
76.110.21TShortStringHelper.ToSingle	1940
76.110.22TShortStringHelper.UpperCase	1940
76.110.23TShortStringHelper.CompareTo	1940
76.110.24TShortStringHelper.Contains	1940
76.110.25TShortStringHelper.CopyTo	1941
76.110.26TShortStringHelper.CountChar	1941
76.110.27TShortStringHelper.DeQuotedString	1941
76.110.28TShortStringHelper.EndsWith	1941
76.110.29TShortStringHelper.GetHashCode	1941
76.110.30TShortStringHelper.IndexOf	1941
76.110.31TShortStringHelper.IndexOfUnQuoted	1942
76.110.32TShortStringHelper.IndexOfAny	1942
76.110.33TShortStringHelper.IndexOfAnyUnquoted	1942
76.110.34TShortStringHelper.Insert	1942
76.110.35TShortStringHelper.IsDelimiter	1943

76.110.36TShortStringHelper.IsEmpty	1943
76.110.37TShortStringHelper.LastDelimiter	1943
76.110.38TShortStringHelper.LastIndexOf	1943
76.110.39TShortStringHelper.LastIndexOfAny	1943
76.110.40TShortStringHelper.PadLeft	1943
76.110.41TShortStringHelper.PadRight	1944
76.110.42TShortStringHelper.QuotedString	1944
76.110.43TShortStringHelper.Remove	1944
76.110.44TShortStringHelper.Replace	1944
76.110.45TShortStringHelper.Split	1944
76.110.46TShortStringHelper.StartsWith	1945
76.110.47TShortStringHelper.Substring	1945
76.110.48TShortStringHelper.ToCharArray	1946
76.110.49TShortStringHelper.ToLower	1946
76.110.50TShortStringHelper.ToLowerInvariant	1946
76.110.51TShortStringHelper.ToUpper	1946
76.110.52TShortStringHelper.ToUpperInvariant	1946
76.110.53TShortStringHelper.Trim	1946
76.110.54TShortStringHelper.TrimLeft	1946
76.110.55TShortStringHelper.TrimRight	1947
76.110.56TShortStringHelper.TrimEnd	1947
76.110.57TShortStringHelper.TrimStart	1947
76.110.58TShortStringHelper.Chars	1947
76.110.59TShortStringHelper.Length	1947
76.111TSimpleRWSync	1947
76.111.1 Description	1947
76.111.2 Interfaces overview	1947
76.111.3 Method overview	1948
76.111.4 TSimpleRWSync.Create	1948
76.111.5 TSimpleRWSync.Destroy	1948
76.111.6 TSimpleRWSync.Beginwrite	1948
76.111.7 TSimpleRWSync.Endwrite	1948
76.111.8 TSimpleRWSync.Beginread	1949
76.111.9 TSimpleRWSync.Endread	1949
76.112TSingleHelper	1949
76.112.1 Description	1949
76.112.2 Method overview	1949
76.112.3 Property overview	1950
76.112.4 TSingleHelper.IsNan	1950
76.112.5 TSingleHelper.IsInfinity	1950

76.112.6	TSingleHelper.IsNegativeInfinity	1950
76.112.7	TSingleHelper.IsPositiveInfinity	1951
76.112.8	TSingleHelper.Parse	1951
76.112.9	TSingleHelper.Size	1951
76.112.10	TSingleHelper.ToString	1951
76.112.11	TSingleHelper.TryParse	1952
76.112.12	TSingleHelper.BuildUp	1952
76.112.13	TSingleHelper.Exponent	1953
76.112.14	TSingleHelper.Fraction	1953
76.112.15	TSingleHelper.Mantissa	1953
76.112.16	TSingleHelper.SpecialType	1953
76.112.17	TSingleHelper.Bytes	1954
76.112.18	TSingleHelper.Words	1954
76.112.19	TSingleHelper.Sign	1954
76.112.20	TSingleHelper.Exp	1954
76.112.21	TSingleHelper.Frac	1955
76.113	TSmallIntHelper	1955
76.113.1	Description	1955
76.113.2	Method overview	1955
76.113.3	Property overview	1955
76.113.4	TSmallIntHelper.Parse	1956
76.113.5	TSmallIntHelper.Size	1956
76.113.6	TSmallIntHelper.ToString	1956
76.113.7	TSmallIntHelper.TryParse	1956
76.113.8	TSmallIntHelper.ToBoolean	1957
76.113.9	TSmallIntHelper.ToBinString	1957
76.113.10	TSmallIntHelper.ToHexString	1957
76.113.11	TSmallIntHelper.ToSingle	1957
76.113.12	TSmallIntHelper.ToDouble	1958
76.113.13	TSmallIntHelper.ToExtended	1958
76.113.14	TSmallIntHelper.SetBit	1958
76.113.15	TSmallIntHelper.ClearBit	1958
76.113.16	TSmallIntHelper.ToggleBit	1959
76.113.17	TSmallIntHelper.TestBit	1959
76.113.18	TSmallIntHelper.Clear	1959
76.113.19	TSmallIntHelper.HighestSetBitPos	1959
76.113.20	TSmallIntHelper.LowestSetBitPos	1960
76.113.21	TSmallIntHelper.SetBitsCount	1960
76.113.22	TSmallIntHelper.Bits	1960
76.113.23	TSmallIntHelper.Nibbles	1961

76.113.24TSmallIntHelper.Bytes	1961
76.114TUnicodeEncoding	1961
76.114.1 Description	1961
76.114.2 Method overview	1961
76.114.3 TUnicodeEncoding.Create	1962
76.114.4 TUnicodeEncoding.Clone	1962
76.114.5 TUnicodeEncoding.GetMaxByteCount	1962
76.114.6 TUnicodeEncoding.GetMaxCharCount	1962
76.114.7 TUnicodeEncoding.GetPreamble	1963
76.115TUNICODESTRINGBUILDER	1963
76.115.1 Description	1963
76.115.2 Method overview	1963
76.115.3 Property overview	1963
76.115.4 TUNICODESTRINGBUILDER.Create	1963
76.115.5 TUNICODESTRINGBUILDER.Append	1964
76.115.6 TUNICODESTRINGBUILDER.AppendFormat	1965
76.115.7 TUNICODESTRINGBUILDER.AppendLine	1965
76.115.8 TUNICODESTRINGBUILDER.Clear	1965
76.115.9 TUNICODESTRINGBUILDER.CopyTo	1966
76.115.10TUNICODESTRINGBUILDER.EnsureCapacity	1966
76.115.11TUNICODESTRINGBUILDER.Equals	1966
76.115.12TUNICODESTRINGBUILDER.GetEnumerator	1966
76.115.13TUNICODESTRINGBUILDER.Insert	1967
76.115.14TUNICODESTRINGBUILDER.Remove	1968
76.115.15TUNICODESTRINGBUILDER.Replace	1968
76.115.16TUNICODESTRINGBUILDER.ToString	1968
76.115.17TUNICODESTRINGBUILDER.Chars	1969
76.115.18TUNICODESTRINGBUILDER.Length	1969
76.115.19TUNICODESTRINGBUILDER.Capacity	1969
76.115.20TUNICODESTRINGBUILDER.MaxCapacity	1969
76.116TUnicodeStringHelper	1971
76.116.1 Method overview	1971
76.116.2 Property overview	1972
76.116.3 TUnicodeStringHelper.Compare	1972
76.116.4 TUnicodeStringHelper.CompareOrdinal	1972
76.116.5 TUnicodeStringHelper.CompareText	1972
76.116.6 TUnicodeStringHelper.Copy	1972
76.116.7 TUnicodeStringHelper.Create	1973
76.116.8 TUnicodeStringHelper.EndsWith	1973
76.116.9 TUnicodeStringHelper.Equals	1973

76.116.10UnicodeStringHelper.Format	1973
76.116.11UnicodeStringHelper.IsNullOrEmpty	1973
76.116.12UnicodeStringHelper.IsNullOrEmpty	1973
76.116.13UnicodeStringHelper.Join	1974
76.116.14UnicodeStringHelper.LowerCase	1974
76.116.15UnicodeStringHelper.Parse	1974
76.116.16UnicodeStringHelper.ToBoolean	1974
76.116.17UnicodeStringHelper.ToDouble	1974
76.116.18UnicodeStringHelper.ToExtended	1975
76.116.19UnicodeStringHelper.ToInt64	1975
76.116.20UnicodeStringHelper.ToInteger	1975
76.116.21UnicodeStringHelper.ToSingle	1975
76.116.22UnicodeStringHelper.UpperCase	1975
76.116.23UnicodeStringHelper.CompareTo	1975
76.116.24UnicodeStringHelper.Contains	1975
76.116.25UnicodeStringHelper.CopyTo	1976
76.116.26UnicodeStringHelper.CountChar	1976
76.116.27UnicodeStringHelper.DeQuotedString	1976
76.116.28UnicodeStringHelper.EndsWith	1976
76.116.29UnicodeStringHelper.GetHashCode	1976
76.116.30UnicodeStringHelper.IndexOf	1976
76.116.31UnicodeStringHelper.IndexOfUnQuoted	1977
76.116.32UnicodeStringHelper.IndexOfAny	1977
76.116.33UnicodeStringHelper.IndexOfAnyUnquoted	1977
76.116.34UnicodeStringHelper.Insert	1977
76.116.35UnicodeStringHelper.IsDelimiter	1978
76.116.36UnicodeStringHelper.IsEmpty	1978
76.116.37UnicodeStringHelper.LastDelimiter	1978
76.116.38UnicodeStringHelper.LastIndexOf	1978
76.116.39UnicodeStringHelper.LastIndexOfAny	1978
76.116.40UnicodeStringHelper.PadLeft	1978
76.116.41UnicodeStringHelper.PadRight	1979
76.116.42UnicodeStringHelper.QuotedString	1979
76.116.43UnicodeStringHelper.Remove	1979
76.116.44UnicodeStringHelper.Replace	1979
76.116.45UnicodeStringHelper.Split	1979
76.116.46UnicodeStringHelper.StartsWith	1980
76.116.47UnicodeStringHelper.Substring	1981
76.116.48UnicodeStringHelper.ToCharArray	1981
76.116.49UnicodeStringHelper.ToLower	1981

76.116.50UnicodeStringHelper.ToLowerInvariant	1981
76.116.51UnicodeStringHelper.ToUpper	1981
76.116.52UnicodeStringHelper.ToUpperInvariant	1981
76.116.53UnicodeStringHelper.Trim	1981
76.116.54UnicodeStringHelper.TrimLeft	1981
76.116.55UnicodeStringHelper.TrimRight	1982
76.116.56UnicodeStringHelper.TrimEnd	1982
76.116.57UnicodeStringHelper.TrimStart	1982
76.116.58UnicodeStringHelper.Chars	1982
76.116.59UnicodeStringHelper.Length	1982
76.117TUTF7Encoding	1982
76.117.1 Description	1982
76.117.2 Method overview	1982
76.117.3 TUTF7Encoding.Create	1983
76.117.4 TUTF7Encoding.Clone	1983
76.117.5 TUTF7Encoding.GetMaxByteCount	1983
76.117.6 TUTF7Encoding.GetMaxCharCount	1983
76.118TUTF8Encoding	1983
76.118.1 Description	1983
76.118.2 Method overview	1984
76.118.3 TUTF8Encoding.Create	1984
76.118.4 TUTF8Encoding.Clone	1984
76.118.5 TUTF8Encoding.GetMaxByteCount	1984
76.118.6 TUTF8Encoding.GetMaxCharCount	1984
76.118.7 TUTF8Encoding.GetPreamble	1985
76.119TWideStringHelper	1987
76.119.1 Method overview	1987
76.119.2 Property overview	1988
76.119.3 TWideStringHelper.Compare	1988
76.119.4 TWideStringHelper.CompareOrdinal	1988
76.119.5 TWideStringHelper.CompareText	1988
76.119.6 TWideStringHelper.Copy	1988
76.119.7 TWideStringHelper.Create	1989
76.119.8 TWideStringHelper.EndsWith	1989
76.119.9 TWideStringHelper.Equals	1989
76.119.10TWideStringHelper.Format	1989
76.119.11TWideStringHelper.IsNullOrEmpty	1989
76.119.12TWideStringHelper.IsNullOrEmptyOrWhiteSpace	1989
76.119.13TWideStringHelper.Join	1990
76.119.14TWideStringHelper.LowerCase	1990

76.119.15TWideStringHelper.Parse	1990
76.119.16TWideStringHelper.ToBoolean	1990
76.119.17TWideStringHelper.ToDouble	1990
76.119.18TWideStringHelper.ToExtended	1991
76.119.19TWideStringHelper.ToInt64	1991
76.119.20TWideStringHelper.ToInteger	1991
76.119.21TWideStringHelper.ToSingle	1991
76.119.22TWideStringHelper.UpperCase	1991
76.119.23TWideStringHelper.CompareTo	1991
76.119.24TWideStringHelper.Contains	1991
76.119.25TWideStringHelper.CopyTo	1992
76.119.26TWideStringHelper.CountChar	1992
76.119.27TWideStringHelper.DeQuotedString	1992
76.119.28TWideStringHelper.EndsWith	1992
76.119.29TWideStringHelper.GetHashCode	1992
76.119.30TWideStringHelper.IndexOf	1992
76.119.31TWideStringHelper.IndexOfUnQuoted	1993
76.119.32TWideStringHelper.IndexOfAny	1993
76.119.33TWideStringHelper.IndexOfAnyUnquoted	1993
76.119.34TWideStringHelper.Insert	1993
76.119.35TWideStringHelper.IsDelimiter	1994
76.119.36TWideStringHelper.IsEmpty	1994
76.119.37TWideStringHelper.LastDelimiter	1994
76.119.38TWideStringHelper.LastIndexOf	1994
76.119.39TWideStringHelper.LastIndexOfAny	1994
76.119.40TWideStringHelper.PadLeft	1994
76.119.41TWideStringHelper.PadRight	1995
76.119.42TWideStringHelper.QuotedString	1995
76.119.43TWideStringHelper.Remove	1995
76.119.44TWideStringHelper.Replace	1995
76.119.45TWideStringHelper.Split	1995
76.119.46TWideStringHelper.StartsWith	1996
76.119.47TWideStringHelper.Substring	1996
76.119.48TWideStringHelper.ToCharArray	1997
76.119.49TWideStringHelper.ToLower	1997
76.119.50TWideStringHelper.ToLowerInvariant	1997
76.119.51TWideStringHelper.ToUpper	1997
76.119.52TWideStringHelper.ToUpperInvariant	1997
76.119.53TWideStringHelper.Trim	1997
76.119.54TWideStringHelper.TrimLeft	1997

76.119.55TWideStringHelper.TrimRight	1998
76.119.56TWideStringHelper.TrimEnd	1998
76.119.57TWideStringHelper.TrimStart	1998
76.119.58TWideStringHelper.Chars	1998
76.119.59TWideStringHelper.Length	1998
76.120TWordBoolHelper	1998
76.120.1 Description	1998
76.120.2 Method overview	1999
76.120.3 TWordBoolHelper.Parse	1999
76.120.4 TWordBoolHelper.Size	1999
76.120.5 TWordBoolHelper.ToString	1999
76.120.6 TWordBoolHelper.TryParse	2000
76.120.7 TWordBoolHelper.ToInteger	2000
76.121TWordHelper	2000
76.121.1 Description	2000
76.121.2 Method overview	2001
76.121.3 Property overview	2001
76.121.4 TWordHelper.Parse	2001
76.121.5 TWordHelper.Size	2001
76.121.6 TWordHelper.ToString	2002
76.121.7 TWordHelper.TryParse	2002
76.121.8 TWordHelper.ToBoolean	2002
76.121.9 TWordHelper.ToDouble	2002
76.121.10TWordHelper.ToExtended	2003
76.121.11TWordHelper.ToBinString	2003
76.121.12TWordHelper.ToHexString	2003
76.121.13TWordHelper.ToSingle	2003
76.121.14TWordHelper.SetBit	2004
76.121.15TWordHelper.ClearBit	2004
76.121.16TWordHelper.ToggleBit	2004
76.121.17TWordHelper.TestBit	2004
76.121.18TWordHelper.Clear	2005
76.121.19TWordHelper.HighestSetBitPos	2005
76.121.20TWordHelper.LowestSetBitPos	2005
76.121.21TWordHelper.SetBitsCount	2005
76.121.22TWordHelper.Bits	2006
76.121.23TWordHelper.Nibbles	2006
76.121.24TWordHelper.Bytes	2006
77 Reference for unit 'Types'	2007

77.1	Used units	2007
77.2	Overview	2007
77.3	Constants, types and variables	2007
77.3.1	Constants	2007
77.3.2	Types	2012
77.4	Procedures and functions	2020
77.4.1	Bounds	2020
77.4.2	CenteredRect	2020
77.4.3	CenterPoint	2020
77.4.4	EqualRect	2020
77.4.5	InflateRect	2021
77.4.6	IntersectRect	2021
77.4.7	IntersectRectF	2021
77.4.8	IsEmptyRect	2021
77.4.9	MinPoint	2022
77.4.10	MultiplyRect	2022
77.4.11	NormalizeRect	2022
77.4.12	NormalizeRectF	2022
77.4.13	OffsetRect	2022
77.4.14	Point	2022
77.4.15	PointF	2023
77.4.16	PtInRect	2023
77.4.17	Rect	2023
77.4.18	RectCenter	2023
77.4.19	RectF	2023
77.4.20	RectHeight	2024
77.4.21	RectWidth	2024
77.4.22	ScalePoint	2024
77.4.23	Size	2024
77.4.24	SplitRect	2024
77.4.25	UnionRect	2024
77.4.26	UnionRectF	2025
77.5	TPoint	2025
77.5.1	Method overview	2026
77.5.2	TPoint.Zero	2026
77.5.3	TPoint.Add	2026
77.5.4	TPoint.Distance	2026
77.5.5	TPoint.IsZero	2027
77.5.6	TPoint.Subtract	2027
77.5.7	TPoint.SetLocation	2027

77.5.8	TPoint.Offset	2027
77.5.9	TPoint.Angle	2027
77.5.10	TPoint.PointInCircle	2027
77.5.11	TPoint.equal(TPoint,TPoint):Boolean	2027
77.5.12	TPoint.notequal(TPoint,TPoint):Boolean	2027
77.5.13	TPoint.add(TPoint,TPoint):TPoint	2028
77.5.14	TPoint.subtract(TPoint,TPoint):TPoint	2028
77.5.15	TPoint.assign(TSmallPoint):TPoint	2028
77.5.16	TPoint.explicit(TPoint):TSmallPoint	2028
77.6	TPoint3D	2028
77.6.1	Method overview	2028
77.6.2	TPoint3D.Create	2029
77.6.3	TPoint3D.Offset	2029
77.7	TPointF	2029
77.7.1	Method overview	2031
77.7.2	TPointF.Add	2031
77.7.3	TPointF.Distance	2031
77.7.4	TPointF.DotProduct	2032
77.7.5	TPointF.IsZero	2032
77.7.6	TPointF.Subtract	2032
77.7.7	TPointF.SetLocation	2032
77.7.8	TPointF.Offset	2032
77.7.9	TPointF.EqualsTo	2032
77.7.10	TPointF.Scale	2032
77.7.11	TPointF.Ceiling	2032
77.7.12	TPointF.Truncate	2033
77.7.13	TPointF.Floor	2033
77.7.14	TPointF.Round	2033
77.7.15	TPointF.Length	2033
77.7.16	TPointF.Rotate	2033
77.7.17	TPointF.Reflect	2033
77.7.18	TPointF.MidPoint	2033
77.7.19	TPointF.PointInCircle	2033
77.7.20	TPointF.Zero	2034
77.7.21	TPointF.Angle	2034
77.7.22	TPointF.AngleCosine	2034
77.7.23	TPointF.CrossProduct	2034
77.7.24	TPointF.Normalize	2034
77.7.25	TPointF.Create	2034
77.7.26	TPointF.equal(TPointF,TPointF):Boolean	2034

77.7.27	TPointF.notequal(TPointF,TPointF):Boolean	2034
77.7.28	TPointF.add(TPointF,TPointF):TPointF	2035
77.7.29	TPointF.subtract(TPointF,TPointF):TPointF	2035
77.7.30	TPointF.negative(TPointF):TPointF	2035
77.7.31	TPointF.multiply(TPointF,TPointF):TPointF	2035
77.7.32	TPointF.multiply(TPointF,single):TPointF	2035
77.7.33	TPointF.multiply(single,TPointF):TPointF	2035
77.7.34	TPointF.divide(TPointF,single):TPointF	2035
77.7.35	TPointF.assign(TPoint):TPointF	2035
77.7.36	TPointF.power(TPointF,TPointF):Single	2036
77.8	TRect	2036
77.8.1	Method overview	2037
77.8.2	Property overview	2037
77.8.3	TRect.Create	2038
77.8.4	TRect.equal(TRect,TRect):Boolean	2038
77.8.5	TRect.notequal(TRect,TRect):Boolean	2038
77.8.6	TRect.add(TRect,TRect):TRect	2038
77.8.7	TRect.multiply(TRect,TRect):TRect	2038
77.8.8	TRect.Empty	2038
77.8.9	TRect.NormalizeRect	2038
77.8.10	TRect.IsEmpty	2038
77.8.11	TRect.Contains	2039
77.8.12	TRect.IntersectsWith	2039
77.8.13	TRect.Intersect	2039
77.8.14	TRect.Union	2039
77.8.15	TRect.Offset	2039
77.8.16	TRect.SetLocation	2039
77.8.17	TRect.Inflate	2039
77.8.18	TRect.CenterPoint	2039
77.8.19	TRect.SplitRect	2040
77.8.20	TRect.Height	2040
77.8.21	TRect.Width	2040
77.8.22	TRect.Size	2040
77.8.23	TRect.Location	2040
77.9	TRectF	2040
77.9.1	Method overview	2042
77.9.2	Property overview	2042
77.9.3	TRectF.Create	2042
77.9.4	TRectF.equal(TRectF,TRectF):Boolean	2043
77.9.5	TRectF.notequal(TRectF,TRectF):Boolean	2043

77.9.6	TRectF.add(TRectF,TRectF):TRectF	2043
77.9.7	TRectF.multiply(TRectF,TRectF):TRectF	2043
77.9.8	TRectF.assign(TRect):TRectF	2043
77.9.9	TRectF.Empty	2043
77.9.10	TRectF.Intersect	2043
77.9.11	TRectF.Union	2044
77.9.12	TRectF.Ceiling	2044
77.9.13	TRectF.CenterAt	2044
77.9.14	TRectF.CenterPoint	2044
77.9.15	TRectF.Contains	2044
77.9.16	TRectF.EqualsTo	2044
77.9.17	TRectF.Fit	2044
77.9.18	TRectF.FitInto	2044
77.9.19	TRectF.IntersectsWith	2045
77.9.20	TRectF.IsEmpty	2045
77.9.21	TRectF.PlaceInto	2045
77.9.22	TRectF.Round	2045
77.9.23	TRectF.SnapToPixel	2045
77.9.24	TRectF.Truncate	2045
77.9.25	TRectF.Inflate	2045
77.9.26	TRectF.Normalize	2045
77.9.27	TRectF.NormalizeRect	2046
77.9.28	TRectF.Offset	2046
77.9.29	TRectF.SetLocation	2046
77.9.30	TRectF.Width	2046
77.9.31	TRectF.Height	2046
77.9.32	TRectF.Size	2046
77.9.33	TRectF.Location	2046
77.10	TSize	2047
77.10.1	Method overview	2047
77.10.2	Property overview	2047
77.10.3	TSize.Add	2047
77.10.4	TSize.Distance	2048
77.10.5	TSize.IsZero	2048
77.10.6	TSize.Subtract	2048
77.10.7	TSize.equal(TSize,TSize):Boolean	2048
77.10.8	TSize.notequal(TSize,TSize):Boolean	2048
77.10.9	TSize.add(TSize,TSize):TSize	2048
77.10.10	TSize.subtract(TSize,TSize):TSize	2048
77.10.11	TSize.Width	2048

77.10.12	TSize.Height	2049
77.11	TSizeF	2049
77.11.1	Method overview	2050
77.11.2	Property overview	2050
77.11.3	TSizeF.Add	2050
77.11.4	TSizeF.Distance	2050
77.11.5	TSizeF.IsZero	2050
77.11.6	TSizeF.Subtract	2051
77.11.7	TSizeF.SwapDimensions	2051
77.11.8	TSizeF.Scale	2051
77.11.9	TSizeF.Ceiling	2051
77.11.10	TSizeF.Truncate	2051
77.11.11	TSizeF.Floor	2051
77.11.12	TSizeF.Round	2051
77.11.13	TSizeF.Length	2051
77.11.14	TSizeF.Create	2052
77.11.15	TSizeF.equal(TSizeF,TSizeF):Boolean	2052
77.11.16	TSizeF.notequal(TSizeF,TSizeF):Boolean	2052
77.11.17	TSizeF.add(TSizeF,TSizeF):TSizeF	2052
77.11.18	TSizeF.subtract(TSizeF,TSizeF):TSizeF	2052
77.11.19	TSizeF.negative(TSizeF):TSizeF	2052
77.11.20	TSizeF.multiply(TSizeF,single):TSizeF	2052
77.11.21	TSizeF.multiply(single,TSizeF):TSizeF	2052
77.11.22	TSizeF.assign(TPointF):TSizeF	2053
77.11.23	TSizeF.assign(TSize):TSizeF	2053
77.11.24	TSizeF.assign(TSizeF):TPointF	2053
77.11.25	TSizeF.Width	2053
77.11.26	TSizeF.Height	2053
77.12	IClassFactory	2053
77.12.1	Description	2053
77.12.2	Method overview	2053
77.12.3	IClassFactory.CreateInstance	2053
77.12.4	IClassFactory.LockServer	2054
77.13	ISequentialStream	2054
77.13.1	Description	2054
77.13.2	Method overview	2054
77.13.3	ISequentialStream.Read	2054
77.13.4	ISequentialStream.Write	2054
77.14	IStream	2055
77.14.1	Description	2055

77.14.2	Method overview	2055
77.14.3	ISStream.Seek	2055
77.14.4	ISStream.SetSize	2055
77.14.5	ISStream.CopyTo	2055
77.14.6	ISStream.Commit	2056
77.14.7	ISStream.Revert	2056
77.14.8	ISStream.LockRegion	2056
77.14.9	ISStream.UnlockRegion	2056
77.14.10	ISStream.Stat	2057
77.14.11	ISStream.Clone	2057
77.15	TBitConverter	2057
77.15.1	Method overview	2057
77.15.2	TBitConverter.UnsafeFrom	2057
77.15.3	TBitConverter.From	2057
77.15.4	TBitConverter.UnsafeInto	2058
77.15.5	TBitConverter.Into	2058
78	Reference for unit 'TypeInfo'	2059
78.1	Used units	2059
78.2	Overview	2059
78.3	Auxiliary functions.	2069
78.4	Getting or setting property values.	2069
78.5	Examining published property information.	2070
78.6	Constants, types and variables	2070
78.6.1	Constants	2070
78.6.2	Types	2073
78.7	Procedures and functions	2081
78.7.1	AddEnumElementAliases	2081
78.7.2	AlignPTypeInfo	2081
78.7.3	AlignTParamFlags	2082
78.7.4	AlignTypeData	2082
78.7.5	DerefTypeInfoPtr	2082
78.7.6	FindPropInfo	2082
78.7.7	GetDynArrayProp	2083
78.7.8	GetEnumeratedAliasValue	2084
78.7.9	GetEnumName	2084
78.7.10	GetEnumNameCount	2085
78.7.11	GetEnumProp	2085
78.7.12	GetEnumValue	2086
78.7.13	GetFloatProp	2086

78.7.14	GetInt64Prop	2087
78.7.15	GetInterfaceProp	2088
78.7.16	GetMethodProp	2088
78.7.17	GetObjectProp	2090
78.7.18	GetObjectPropClass	2091
78.7.19	GetOrdProp	2092
78.7.20	GetPropInfo	2093
78.7.21	GetPropInfos	2093
78.7.22	GetPropList	2094
78.7.23	GetPropValue	2095
78.7.24	GetRawByteStrProp	2096
78.7.25	GetRawInterfaceProp	2096
78.7.26	GetSetProp	2097
78.7.27	GetStrProp	2098
78.7.28	GetTypeData	2099
78.7.29	GetUnicodeStrProp	2099
78.7.30	GetVariantProp	2099
78.7.31	GetWideStrProp	2100
78.7.32	IsPublishedProp	2100
78.7.33	IsReadableProp	2101
78.7.34	IsStoredProp	2101
78.7.35	IsWriteableProp	2102
78.7.36	PropIsType	2102
78.7.37	PropType	2103
78.7.38	RemoveEnumElementAliases	2104
78.7.39	SetDynArrayProp	2104
78.7.40	SetEnumProp	2105
78.7.41	SetFloatProp	2105
78.7.42	SetInt64Prop	2105
78.7.43	SetInterfaceProp	2106
78.7.44	SetMethodProp	2106
78.7.45	SetObjectProp	2107
78.7.46	SetOrdProp	2107
78.7.47	SetPropValue	2108
78.7.48	SetRawByteStrProp	2108
78.7.49	SetRawInterfaceProp	2108
78.7.50	SetSetProp	2109
78.7.51	SetStrProp	2109
78.7.52	SetToString	2109
78.7.53	SetUnicodeStrProp	2110

78.7.54	SetVariantProp	2111
78.7.55	SetWideStrProp	2111
78.7.56	StringToSet	2111
78.8	TArrayTypeData	2112
78.8.1	Property overview	2112
78.8.2	TArrayTypeData.ElType	2113
78.8.3	TArrayTypeData.Dims	2113
78.9	TClassData	2113
78.9.1	Property overview	2113
78.9.2	TClassData.UnitName	2113
78.9.3	TClassData.PropertyTable	2114
78.10	TInterfaceData	2114
78.10.1	Property overview	2114
78.10.2	TInterfaceData.UnitName	2115
78.10.3	TInterfaceData.PropertyTable	2115
78.10.4	TInterfaceData.MethodTable	2115
78.11	TInterfaceRawData	2115
78.11.1	Property overview	2116
78.11.2	TInterfaceRawData.UnitName	2116
78.11.3	TInterfaceRawData.IIDStr	2116
78.11.4	TInterfaceRawData.PropertyTable	2116
78.11.5	TInterfaceRawData.MethodTable	2117
78.12	TIntfMethodEntry	2117
78.12.1	Property overview	2118
78.12.2	TIntfMethodEntry.Name	2118
78.12.3	TIntfMethodEntry.Param	2118
78.12.4	TIntfMethodEntry.ResultLocs	2118
78.12.5	TIntfMethodEntry.Tail	2119
78.12.6	TIntfMethodEntry.Next	2119
78.13	TIntfMethodTable	2119
78.13.1	Property overview	2119
78.13.2	TIntfMethodTable.Method	2120
78.14	TManagedField	2120
78.14.1	Property overview	2120
78.14.2	TManagedField.TypeRef	2120
78.15	TParameterLocation	2120
78.15.1	Property overview	2121
78.15.2	TParameterLocation.Reference	2121
78.15.3	TParameterLocation.RegType	2122
78.15.4	TParameterLocation.ShiftVal	2122

78.16	TParameterLocations	2122
78.16.1	Property overview	2122
78.16.2	TParameterLocations.Location	2123
78.16.3	TParameterLocations.Tail	2123
78.17	TProcedureParam	2123
78.17.1	Property overview	2123
78.17.2	TProcedureParam.ParamType	2124
78.17.3	TProcedureParam.Flags	2124
78.18	TProcedureSignature	2124
78.18.1	Method overview	2124
78.18.2	Property overview	2124
78.18.3	TProcedureSignature.GetParam	2125
78.18.4	TProcedureSignature.ResultType	2125
78.19	TPropData	2125
78.19.1	Property overview	2125
78.19.2	TPropData.Prop	2126
78.19.3	TPropData.Tail	2126
78.20	TPropInfo	2126
78.20.1	Property overview	2127
78.20.2	TPropInfo.PropType	2127
78.20.3	TPropInfo.Tail	2127
78.20.4	TPropInfo.Next	2127
78.21	TTypeData	2128
78.21.1	Property overview	2143
78.21.2	TTypeData.BaseType	2143
78.21.3	TTypeData.CompType	2143
78.21.4	TTypeData.ParentInfo	2143
78.21.5	TTypeData.RecInitData	2144
78.21.6	TTypeData.HelperParent	2144
78.21.7	TTypeData.ExtendedInfo	2144
78.21.8	TTypeData.IntfParent	2144
78.21.9	TTypeData.RawIntfParent	2145
78.21.10	TTypeData.IIDStr	2145
78.21.11	TTypeData.ElType2	2145
78.21.12	TTypeData.ElType	2145
78.21.13	TTypeData.InstanceType	2145
78.21.14	TTypeData.RefType	2145
78.22	TVmtFieldEntry	2146
78.22.1	Property overview	2146
78.22.2	TVmtFieldEntry.Tail	2146

78.22.3	TVmtFieldEntry.Next	2146
78.23	TVmtFieldTable	2147
78.23.1	Property overview	2147
78.23.2	TVmtFieldTable.Field	2147
78.24	TVmtMethodParam	2147
78.24.1	Property overview	2148
78.24.2	TVmtMethodParam.Name	2148
78.24.3	TVmtMethodParam.Tail	2148
78.24.4	TVmtMethodParam.Next	2148
78.25	TVmtMethodTable	2149
78.25.1	Property overview	2149
78.25.2	TVmtMethodTable.Entry	2149
78.26	EPropertyConvertError	2149
78.26.1	Description	2149
78.27	EPropertyError	2149
78.27.1	Description	2149
79	Reference for unit 'unicodedata'	2150
79.1	Used units	2150
79.2	Overview	2150
79.3	Constants, types and variables	2151
79.3.1	Resource strings	2151
79.3.2	Constants	2151
79.3.3	Types	2155
79.4	Procedures and functions	2157
79.4.1	AddAliasCollation	2157
79.4.2	BytesToName	2157
79.4.3	BytesToString	2157
79.4.4	CanonicalOrder	2157
79.4.5	CompareSortKey	2158
79.4.6	ComputeSortKey	2158
79.4.7	FilterString	2159
79.4.8	FindCollation	2159
79.4.9	FreeCollation	2159
79.4.10	FromUCS4	2159
79.4.11	GetCollationCount	2160
79.4.12	GetProps	2160
79.4.13	GetPropUCA	2160
79.4.14	IncrementalCompareString	2160
79.4.15	LoadCollation	2161

79.4.16	NormalizeNFD	2162
79.4.17	PrepareCollation	2162
79.4.18	RegisterCollation	2162
79.4.19	ToUCS4	2163
79.4.20	UnicodeIsHighSurrogate	2163
79.4.21	UnicodeIsLowSurrogate	2163
79.4.22	UnicodeIsSurrogatePair	2164
79.4.23	UnicodeToLower	2164
79.4.24	UnicodeToUpper	2164
79.4.25	UnregisterCollation	2165
79.4.26	UnregisterCollations	2165
79.5	TCollationTable	2165
79.5.1	Method overview	2166
79.5.2	Property overview	2166
79.5.3	TCollationTable.NormalizeName	2166
79.5.4	TCollationTable.Clear	2166
79.5.5	TCollationTable.IndexOf	2166
79.5.6	TCollationTable.Find	2166
79.5.7	TCollationTable.Add	2167
79.5.8	TCollationTable.Remove	2167
79.5.9	TCollationTable.Item	2167
79.5.10	TCollationTable.Count	2167
79.5.11	TCollationTable.Capacity	2167
79.6	TUCA_DataBook	2167
79.6.1	Method overview	2168
79.6.2	TUCA_DataBook.IsVariable	2168
79.7	TUCA_PropItemContextRec	2168
79.7.1	Method overview	2169
79.7.2	TUCA_PropItemContextRec.GetCodePoints	2169
79.7.3	TUCA_PropItemContextRec.GetWeights	2169
79.8	TUCA_PropItemContextTreeNodeRec	2169
79.8.1	Method overview	2169
79.8.2	TUCA_PropItemContextTreeNodeRec.GetLeftNode	2169
79.8.3	TUCA_PropItemContextTreeNodeRec.GetRightNode	2170
79.9	TUCA_PropItemContextTreeRec	2170
79.9.1	Method overview	2170
79.9.2	Property overview	2170
79.9.3	TUCA_PropItemContextTreeRec.GetData	2170
79.9.4	TUCA_PropItemContextTreeRec.Find	2170
79.9.5	TUCA_PropItemContextTreeRec.Data	2171

79.10	TUCA_PropItemRec	2171
79.10.1	Method overview	2172
79.10.2	Property overview	2172
79.10.3	TUCA_PropItemRec.HasCodePoint	2172
79.10.4	TUCA_PropItemRec.IsValid	2172
79.10.5	TUCA_PropItemRec.GetWeightArray	2172
79.10.6	TUCA_PropItemRec.GetSelfOnlySize	2172
79.10.7	TUCA_PropItemRec.GetContextual	2173
79.10.8	TUCA_PropItemRec.GetContext	2173
79.10.9	TUCA_PropItemRec.IsDeleted	2173
79.10.10	TUCA_PropItemRec.IsWeightCompress_1	2173
79.10.11	TUCA_PropItemRec.IsWeightCompress_2	2173
79.10.12	TUCA_PropItemRec.CodePoint	2173
79.10.13	TUCA_PropItemRec.Contextual	2174
79.11	TUC_Prop	2174
79.11.1	Property overview	2175
79.11.2	TUC_Prop.CategoryData	2175
79.11.3	TUC_Prop.NumericIndex	2175
79.11.4	TUC_Prop.SimpleUpperCase	2175
79.11.5	TUC_Prop.SimpleLowerCase	2175
79.11.6	TUC_Prop.DecompositionID	2176
79.11.7	TUC_Prop.Category	2176
79.11.8	TUC_Prop.WhiteSpace	2176
79.11.9	TUC_Prop.HangulSyllable	2176
79.11.10	TUC_Prop.UnifiedIdeograph	2176
79.11.11	TUC_Prop.NumericValue	2177
79.12	TUInt24Rec	2177
79.12.1	Method overview	2180
79.12.2	Property overview	2181
79.12.3	TUInt24Rec.implicit(TUInt24Rec):Cardinal	2181
79.12.4	TUInt24Rec.implicit(TUInt24Rec):LongInt	2181
79.12.5	TUInt24Rec.implicit(TUInt24Rec):Word	2181
79.12.6	TUInt24Rec.implicit(TUInt24Rec):Byte	2182
79.12.7	TUInt24Rec.implicit(Cardinal):TUInt24Rec	2182
79.12.8	TUInt24Rec.equal(TUInt24Rec,TUInt24Rec):Boolean	2182
79.12.9	TUInt24Rec.equal(TUInt24Rec,Cardinal):Boolean	2182
79.12.10	TUInt24Rec.equal(Cardinal,TUInt24Rec):Boolean	2183
79.12.11	TUInt24Rec.equal(TUInt24Rec,LongInt):Boolean	2183
79.12.12	TUInt24Rec.equal(LongInt,TUInt24Rec):Boolean	2183
79.12.13	TUInt24Rec.equal(TUInt24Rec,Word):Boolean	2183

79.12.14	TUInt24Rec.equal(Word,TUInt24Rec):Boolean	2183
79.12.15	TUInt24Rec.equal(TUInt24Rec,Byte):Boolean	2184
79.12.16	TUInt24Rec.equal(Byte,TUInt24Rec):Boolean	2184
79.12.17	TUInt24Rec.notequal(TUInt24Rec,TUInt24Rec):Boolean	2184
79.12.18	TUInt24Rec.notequal(TUInt24Rec,Cardinal):Boolean	2184
79.12.19	TUInt24Rec.notequal(Cardinal,TUInt24Rec):Boolean	2184
79.12.20	TUInt24Rec.greaterthan(TUInt24Rec,TUInt24Rec):Boolean	2185
79.12.21	TUInt24Rec.greaterthan(TUInt24Rec,Cardinal):Boolean	2185
79.12.22	TUInt24Rec.greaterthan(Cardinal,TUInt24Rec):Boolean	2185
79.12.23	TUInt24Rec.greaterthanorequal(TUInt24Rec,TUInt24Rec):Boolean	2185
79.12.24	TUInt24Rec.greaterthanorequal(TUInt24Rec,Cardinal):Boolean	2185
79.12.25	TUInt24Rec.greaterthanorequal(Cardinal,TUInt24Rec):Boolean	2186
79.12.26	TUInt24Rec.less than(TUInt24Rec,TUInt24Rec):Boolean	2186
79.12.27	TUInt24Rec.less than(TUInt24Rec,Cardinal):Boolean	2186
79.12.28	TUInt24Rec.less than(Cardinal,TUInt24Rec):Boolean	2186
79.12.29	TUInt24Rec.less thanorequal(TUInt24Rec,TUInt24Rec):Boolean	2186
79.12.30	TUInt24Rec.less thanorequal(TUInt24Rec,Cardinal):Boolean	2187
79.12.31	TUInt24Rec.less thanorequal(Cardinal,TUInt24Rec):Boolean	2187
79.12.32	TUInt24Rec.byte0	2187
79.12.33	TUInt24Rec.byte1	2187
79.12.34	TUInt24Rec.byte2	2187
80	Reference for unit 'unicodeducet'	2188
80.1	Used units	2188
80.2	Overview	2188
81	Reference for unit 'Unix'	2189
81.1	Used units	2189
81.2	Constants, types and variables	2189
81.2.1	Constants	2189
81.2.2	Types	2195
81.3	Procedures and functions	2204
81.3.1	AssignPipe	2204
81.3.2	AssignStream	2205
81.3.3	EpochToLocal	2207
81.3.4	EpochToUniversal	2207
81.3.5	FpExecL	2207
81.3.6	FpExecLE	2208
81.3.7	FpExecLP	2209
81.3.8	FpExecLPE	2210
81.3.9	FpExecV	2210

81.3.10	FpExecVP	2211
81.3.11	FpExecVPE	2212
81.3.12	fpFlock	2213
81.3.13	fpfStatFS	2213
81.3.14	fpfsync	2213
81.3.15	fpgettimeofday	2214
81.3.16	fpStatFS	2214
81.3.17	fpSystem	2214
81.3.18	FSearch	2215
81.3.19	GetDomainName	2216
81.3.20	GetHostName	2216
81.3.21	GetLocalTimezone	2217
81.3.22	GetTimezoneFile	2217
81.3.23	Gettzdaylight	2217
81.3.24	GetTZInfo	2217
81.3.25	GetTZInfoEx	2218
81.3.26	Gettzname	2218
81.3.27	GetTzseconds	2218
81.3.28	GregorianToJulian	2218
81.3.29	JulianToGregorian	2218
81.3.30	LocalToEpoch	2218
81.3.31	PClose	2218
81.3.32	POpen	2219
81.3.33	ReadTimezoneFile	2220
81.3.34	RefreshTZInfo	2220
81.3.35	ReReadLocalTime	2220
81.3.36	SeekDir	2220
81.3.37	SetTZInfo	2221
81.3.38	TellDir	2221
81.3.39	UniversalToEpoch	2221
81.3.40	WaitProcess	2221
81.3.41	WIFSTOPPED	2222
81.3.42	W_EXITCODE	2222
81.3.43	W_STOPCODE	2222
81.4	TTZInfo	2222
81.5	TTZInfoEx	2223
82	Reference for unit 'unixcp'	2224
82.1	Used units	2224
82.2	Overview	2224

82.3	Constants, types and variables	2224
82.3.1	Constants	2224
82.3.2	Types	2227
82.4	Procedures and functions	2227
82.4.1	GetCodepageByName	2227
82.4.2	GetCodepageData	2228
82.4.3	GetSystemCodepage	2228
82.5	TUnixCpData	2228
83	Reference for unit 'unixtype'	2229
83.1	Used units	2229
83.2	Overview	2229
83.3	Constants, types and variables	2229
83.3.1	Constants	2229
83.3.2	Types	2231
83.4	mbstate_t	2241
83.5	pthread_attr_t	2241
83.6	pthread_condattr_t	2241
83.7	pthread_cond_t	2242
83.8	pthread_mutexattr_t	2242
83.9	pthread_rwlockattr_t	2242
83.10	sched_param	2242
83.11	sem_t	2242
83.12	timespec	2243
83.13	timeval	2243
83.14	TStatfs	2243
83.15	_pthread_fastlock	2244
84	Reference for unit 'unixutil'	2245
84.1	Used units	2245
84.2	Overview	2245
84.3	Procedures and functions	2245
84.3.1	ArrayStringToPPchar	2245
84.3.2	StringToPPChar	2246
85	Reference for unit 'Variants'	2247
85.1	Used units	2247
85.2	Overview	2247
85.3	Constants, types and variables	2247
85.3.1	Constants	2247
85.3.2	Types	2248

85.3.3	Variables	2250
85.4	Procedures and functions	2252
85.4.1	DynArrayFromVariant	2252
85.4.2	DynArrayToVariant	2252
85.4.3	FindCustomVariantType	2252
85.4.4	FindVarData	2253
85.4.5	GetPropValue	2253
85.4.6	GetVariantProp	2253
85.4.7	HandleConversionException	2254
85.4.8	Null	2254
85.4.9	SetClearVarToEmptyParam	2254
85.4.10	SetPropValue	2254
85.4.11	SetVariantProp	2255
85.4.12	Unassigned	2255
85.4.13	VarArrayAsPSafeArray	2255
85.4.14	VarArrayCreate	2255
85.4.15	VarArrayCreateError	2256
85.4.16	VarArrayDimCount	2256
85.4.17	VarArrayHighBound	2257
85.4.18	VarArrayLock	2257
85.4.19	VarArrayLockedError	2257
85.4.20	VarArrayLowBound	2258
85.4.21	VarArrayOf	2258
85.4.22	VarArrayRef	2258
85.4.23	VarArrayUnlock	2258
85.4.24	VarAsError	2259
85.4.25	VarAsType	2259
85.4.26	VarBadIndexError	2259
85.4.27	VarBadTypeError	2259
85.4.28	VarCastError	2260
85.4.29	VarCastErrorOle	2260
85.4.30	VarCheckEmpty	2260
85.4.31	VarClear	2260
85.4.32	VarCompareValue	2261
85.4.33	VarCopyNoInd	2261
85.4.34	VarEnsureRange	2261
85.4.35	VarFromDateTime	2261
85.4.36	VarInRange	2262
85.4.37	VarInvalidArgError	2262
85.4.38	VarInvalidNullOp	2262

85.4.39	VarInvalidOp	2262
85.4.40	VarIsArray	2263
85.4.41	VarIsBool	2263
85.4.42	VarIsByRef	2263
85.4.43	VarIsClear	2263
85.4.44	VarIsCustom	2264
85.4.45	VarIsEmpty	2264
85.4.46	VarIsEmptyParam	2264
85.4.47	VarIsError	2264
85.4.48	VarIsFloat	2265
85.4.49	VarIsNull	2265
85.4.50	VarIsNumeric	2265
85.4.51	VarIsOrdinal	2265
85.4.52	VarIsStr	2266
85.4.53	VarIsType	2266
85.4.54	VarNotImplError	2266
85.4.55	VarOutOfMemoryError	2266
85.4.56	VarOverflowError	2267
85.4.57	VarRangeCheckError	2267
85.4.58	VarResultCheck	2267
85.4.59	VarSameValue	2267
85.4.60	VarSupports	2268
85.4.61	VarToDateTime	2268
85.4.62	VarToStr	2268
85.4.63	VarToStrDef	2268
85.4.64	VarToUnicodeStr	2269
85.4.65	VarToUnicodeStrDef	2269
85.4.66	VarToWideStr	2269
85.4.67	VarToWideStrDef	2269
85.4.68	VarType	2270
85.4.69	VarTypeAsText	2270
85.4.70	VarTypeDeRef	2270
85.4.71	VarTypeIsValidArrayType	2270
85.4.72	VarTypeIsValidElementType	2271
85.4.73	VarUnexpectedError	2271
85.5	EVariantArrayCreateError	2271
85.5.1	Description	2271
85.6	EVariantArrayLockedError	2271
85.6.1	Description	2271
85.7	EVariantBadIndexError	2272

85.7.1	Description	2272
85.8	EVariantBadVarTypeError	2272
85.8.1	Description	2272
85.9	EVariantDispatchError	2272
85.9.1	Description	2272
85.10	EVariantInvalidArgError	2272
85.10.1	Description	2272
85.11	EVariantInvalidNullOpError	2272
85.11.1	Description	2272
85.12	EVariantInvalidOpError	2272
85.12.1	Description	2272
85.13	EVariantNotAnArrayError	2273
85.13.1	Description	2273
85.14	EVariantNotImplError	2273
85.14.1	Description	2273
85.15	EVariantOutOfMemoryError	2273
85.15.1	Description	2273
85.16	EVariantOverflowError	2273
85.16.1	Description	2273
85.17	EVariantParamNotFoundError	2273
85.17.1	Description	2273
85.18	EVariantRangeCheckError	2273
85.18.1	Description	2273
85.19	EVariantTypeCastError	2274
85.19.1	Description	2274
85.20	EVariantUnexpectedError	2274
85.20.1	Description	2274
85.21	IVarInstanceReference	2274
85.21.1	Description	2274
85.21.2	Method overview	2274
85.21.3	IVarInstanceReference.GetInstance	2274
85.22	IVarInvokeable	2275
85.22.1	Description	2275
85.22.2	Method overview	2275
85.22.3	IVarInvokeable.DoFunction	2275
85.22.4	IVarInvokeable.DoProcedure	2275
85.22.5	IVarInvokeable.GetProperty	2276
85.22.6	IVarInvokeable.SetProperty	2276
85.23	TCustomVariantType	2276
85.23.1	Description	2276

85.23.2	Interfaces overview	2277
85.23.3	Method overview	2277
85.23.4	Property overview	2277
85.23.5	TCustomVariantType.Create	2277
85.23.6	TCustomVariantType.Destroy	2277
85.23.7	TCustomVariantType.IsClear	2278
85.23.8	TCustomVariantType.Cast	2278
85.23.9	TCustomVariantType.CastTo	2278
85.23.10	TCustomVariantType.CastToOle	2278
85.23.11	TCustomVariantType.Clear	2279
85.23.12	TCustomVariantType.Copy	2279
85.23.13	TCustomVariantType.BinaryOp	2279
85.23.14	TCustomVariantType.UnaryOp	2280
85.23.15	TCustomVariantType.CompareOp	2280
85.23.16	TCustomVariantType.Compare	2280
85.23.17	TCustomVariantType.VarType	2281
85.24	TInvokeableVariantType	2281
85.24.1	Description	2281
85.24.2	Interfaces overview	2281
85.24.3	Method overview	2281
85.24.4	TInvokeableVariantType.DoFunction	2281
85.24.5	TInvokeableVariantType.DoProcedure	2282
85.24.6	TInvokeableVariantType.GetProperty	2282
85.24.7	TInvokeableVariantType.SetProperty	2282
85.25	TPublishableVariantType	2283
85.25.1	Description	2283
85.25.2	Interfaces overview	2283
85.25.3	Method overview	2283
85.25.4	TPublishableVariantType.GetProperty	2283
85.25.5	TPublishableVariantType.SetProperty	2283
86	Reference for unit 'video'	2284
86.1	Used units	2284
86.2	Overview	2284
86.3	Examples utility unit.	2285
86.4	Writing a custom video driver.	2286
86.5	Constants, types and variables	2289
86.5.1	Constants	2289
86.5.2	Types	2293
86.5.3	Variables	2294

86.6	Procedures and functions	2295
86.6.1	ClearScreen	2295
86.6.2	DefaultErrorHandler	2296
86.6.3	DoneVideo	2296
86.6.4	GetCapabilities	2296
86.6.5	GetCursorType	2297
86.6.6	GetLockScreenCount	2298
86.6.7	GetVideoDriver	2299
86.6.8	GetVideoMode	2299
86.6.9	GetVideoModeCount	2300
86.6.10	GetVideoModeData	2301
86.6.11	InitVideo	2301
86.6.12	LockScreenUpdate	2302
86.6.13	SetCursorPos	2302
86.6.14	SetCursorType	2303
86.6.15	SetVideoDriver	2304
86.6.16	SetVideoMode	2304
86.6.17	UnlockScreenUpdate	2304
86.6.18	UpdateScreen	2305
86.7	TVideoDriver	2305
86.8	TVideoMode	2306
87	Reference for unit 'WinCRT'	2307
87.1	Used units	2307
87.2	Overview	2307
87.3	Constants, types and variables	2307
87.3.1	Variables	2307
87.4	Procedures and functions	2307
87.4.1	delay	2307
87.4.2	keypressed	2308
87.4.3	nosound	2308
87.4.4	readkey	2308
87.4.5	sound	2308
87.4.6	textmode	2309
88	Reference for unit 'WinDirs'	2310
88.1	Used units	2310
88.2	Overview	2310
88.3	Constants, types and variables	2310
88.3.1	Constants	2310
88.4	Procedures and functions	2324

88.4.1	ConvertCSIDLtoFOLDERID	2324
88.4.2	ConvertFOLDERIDtoCSIDL	2324
88.4.3	GetWindowsSpecialDir	2324
88.4.4	GetWindowsSpecialDirUnicode	2324
88.4.5	GetWindowsSystemDirectory	2324
88.4.6	GetWindowsSystemDirectoryUnicode	2325
89	Reference for unit 'x86'	2326
89.1	Used units	2326
89.2	Overview	2326
89.3	Procedures and functions	2326
89.3.1	fpIOperm	2326
89.3.2	fpIoPL	2327
89.3.3	ReadPort	2327
89.3.4	ReadPortB	2327
89.3.5	ReadPortL	2328
89.3.6	ReadPortW	2328
89.3.7	WritePort	2328
89.3.8	WritePortB	2329
89.3.9	WritePortL	2329
89.3.10	WritePortW	2329

About this guide

This document describes all constants, types, variables, functions and procedures as they are declared in the units that come standard with the Free Pascal Run-Time library (RTL).

Throughout this document, we will refer to functions, types and variables with `typewriter` font. Functions and procedures have their own subsections, and for each function or procedure we have the following topics:

Declaration The exact declaration of the function.

Description What does the procedure exactly do ?

Errors What errors can occur.

See Also Cross references to other related functions/commands.

0.1 Overview

The Run-Time Library is the basis of all Free Pascal programs. It contains the basic units that most programs will use, and are made available on all platforms supported by Free pascal (well, more or less).

There are units for compatibility with the Turbo Pascal Run-Time library, and there are units for compatibility with Delphi.

On top of these two sets, there are also a series of units to handle keyboard/mouse and text screens in a cross-platform way.

Other units include platform specific units that implement the specifics of a platform, these are usually needed to support the Turbo Pascal or Delphi units.

Units that fall outside the above outline do not belong in the RTL, but should be included in the packages, or in the FCL.

Chapter 1

Reference for unit 'BaseUnix'

1.1 Used units

Table 1.1: Used units by unit 'BaseUnix'

Name	Page
System	1335
unixtype	2229

1.2 Overview

The BaseUnix unit was implemented by Marco Van de Voort. It contains basic Unix functionality. It supersedes the Linux unit of version 1.0.X of the compiler, but only implements a cleaned up, portable subset of that unit.

For porting FPC to new Unix-like platforms, it should be sufficient to implement the functionality in this unit for the new platform.

1.3 Constants, types and variables

1.3.1 Constants

`ARG_MAX = UnixType.ARG_MAX`

Maximum number of arguments to a program.

`AT_EMPTY_PATH = $1000`

Create link to old directy if path is empty.

`AT_FDCWD = - 100`

Flag for various *at calls to indicate current working directory.

`AT_NO_AUTOMOUNT = $800`

Do not auto mount filesystem if the base dir is a mount point.

AT_RECURSIVE = \$8000

Recurse subdirectories when applying changes.

AT_REMOVEDIR = \$200

Unlink at: specify rmdir behaviour.

AT_STATX_DONT_SYNC = \$4000

Do not synchronize the attributes with the server (if any).

AT_STATX_FORCE_SYNC = \$2000

Force the attributes to be synchronized with the server (if any).

AT_STATX_SYNC_AS_STAT = \$0000

Behave as fpStat.

AT_STATX_SYNC_TYPE = \$6000

Type of synchronisation required from statx.

AT_SYMLINK_FOLLOW = \$400

Dereference symbolic links.

AT_SYMLINK_NOFOLLOW = \$100

Do not dereference symbolic links.

BITSINWORD = 8 * sizeof(cuLong)

Number of bits in a word.

clone_flags_fork = \$01200011

Request fork behaviour for clone call.

ESysE2BIG = 7

System error: Argument list too long.

ESysEACCES = 13

System error: Permission denied.

ESysEADDRINUSE = 98

System error: Address already in use.

ESysEADDRNOTAVAIL = 99

System error: Cannot assign requested address.

ESysEADV = 68

System error: Advertise error.

ESysEAFNOSUPPORT = 97

System error: Address family not supported by protocol.

ESysEAGAIN = 11

System error: Try again.

ESysEALREADY = 114

System error: Operation already in progress.

ESysEBADE = 52

System error: Invalid exchange.

ESysEBADF = 9

System error: Bad file number.

ESysEBADFD = 77

System error: File descriptor in bad state.

ESysEBADMSG = 74

System error: Not a data message.

ESysEBADR = 53

System error: Invalid request descriptor.

ESysEBADRQC = 56

System error: Invalid request code.

ESysEBADSLT = 57

System error: Invalid slot.

ESysEBFONT = 59

System error: Bad font file format.

ESysEBUSY = 16

System error: Device or resource busy.

ESysECANCELED = 125

Operation canceled.

ESysECHILD = 10

System error: No child processes.

ESysECHRNG = 44

System error: Channel number out of range.

ESysECOMM = 70

System error: Communication error on send.

ESysECONNABORTED = 103

System error: Software caused connection abort.

ESysECONNREFUSED = 111

System error: Connection refused.

ESysECONNRESET = 104

System error: Connection reset by peer.

ESysEDEADLK = 35

System error: Resource deadlock would occur.

ESysEDEADLOCK = ESysEDEADLK

System error: File locking deadlock error.

ESysEDESTADDRREQ = 89

System error: Destination address required.

ESysEDOM = 33

System error: Math argument out of domain of func.

ESysEDOTDOT = 73

System error: RFS specific error.

ESysEDQUOT = 122

System error: Quota exceeded.

ESysEEXIST = 17

System error: File exists.

ESysEFAULT = 14

System error: Bad address.

ESysEFBIG = 27

System error: File too large.

ESysEHOSTDOWN = 112

System error: Host is down.

ESysEHOSTUNREACH = 113

System error: No route to host.

ESysEIDRM = 43

System error: Identifier removed.

ESysEILSEQ = 84

System error: Illegal byte sequence.

ESysEINPROGRESS = 115

System error: Operation now in progress.

ESysEINTR = 4

System error: Interrupted system call.

ESysEINVAL = 22

System error: Invalid argument.

ESysEIO = 5

System error: I/O error.

ESysEISCONN = 106

System error: Transport endpoint is already connected.

ESysEISDIR = 21

System error: Is a directory.

ESysEISNAM = 120

System error: Is a named type file.

ESysEKEYEXPIRED = 127

Key has expired (Linux kernel module).

ESysEKEYREJECTED = 129

Key was rejected by service (Linux kernel module).

ESysEKEYREVOKED = 128

Key has been revoked (Linux kernel module).

ESysEL2HLT = 51

System error: Level 2 halted.

ESysEL2NSYNC = 45

System error: Level 2 not synchronized.

ESysEL3HLT = 46

System error: Level 3 halted.

ESysEL3RST = 47

System error: Level 3 reset.

ESysELIBACC = 79

System error: Can not access a needed shared library.

ESysELIBBAD = 80

System error: Accessing a corrupted shared library.

ESysELIBEXEC = 83

System error: Cannot exec a shared library directly.

ESysELIBMAX = 82

System error: Attempting to link in too many shared libraries.

ESysELIBSCN = 81

System error: .lib section in a.out corrupted.

ESysELNRNG = 48

System error: Link number out of range.

ESysELOOP = 40

System error: Too many symbolic links encountered.

ESysEMEDIUMTYPE = 124

Wrong medium type.

ESysEMFILE = 24

System error: Too many open files.

ESysEMLINK = 31

System error: Too many links.

ESysEMSGSIZE = 90

System error: Message too long.

ESysEMULTIHOP = 72

System error: Multihop attempted.

ESysENAMETOOLONG = 36

System error: File name too long.

ESysENAVAIL = 119

System error: No XENIX semaphores available.

ESysENETDOWN = 100

System error: Network is down.

ESysENETRESET = 102

System error: Network dropped connection because of reset.

ESysENETUNREACH = 101

System error: Network is unreachable.

ESysENFILE = 23

System error: File table overflow.

ESysENOANO = 55

System error: No anode.

ESysENOBUFFS = 105

System error: No buffer space available.

ESysENOC SI = 50

System error: No CSI structure available.

ESysENODATA = 61

System error: No data available.

ESysENODEV = 19

System error: No such device.

ESysENOENT = 2

System error: No such file or directory.

ESysENOEXEC = 8

System error: Exec format error.

ESysENOKEY = 126

Required key not available (Linux kernel module).

ESysENOLCK = 37

System error: No record locks available.

ESysENOLINK = 67

System error: Link has been severed.

ESysENOMEDIUM = 123

No medium present.

ESysENOMEM = 12

System error: Out of memory.

ESysENOMSG = 42

System error: No message of desired type.

ESysENONET = 64

System error: Machine is not on the network.

ESysENOPKG = 65

System error: Package not installed.

ESysENOPROTOPT = 92

System error: Protocol not available.

ESysENOSPC = 28

System error: No space left on device.

ESysENOSR = 63

System error: Out of streams resources.

ESysENOSTR = 60

System error: Device not a stream.

ESysENOSYS = 38

System error: Function not implemented.

ESysENOTBLK = 15

System error: Block device required.

ESysENOTCONN = 107

System error: Transport endpoint is not connected.

ESysENOTDIR = 20

System error: Not a directory.

ESysENOTEMPTY = 39

System error: Directory not empty.

ESysENOTNAM = 118

System error: Not a XENIX named type file.

ESysENOTRECOVERABLE = 131

State not recoverable (mutexes).

ESysENOTSOCK = 88

System error: Socket operation on non-socket.

ESysENOTTY = 25

System error: Not a typewriter.

ESysENOTUNIQ = 76

System error: Name not unique on network.

ESysENXIO = 6

System error: No such device or address.

ESysEOPNOTSUPP = 95

System error: Operation not supported on transport endpoint.

ESysEOVERFLOW = 75

System error: Value too large for defined data type.

ESysEOWNERDEAD = 130

Owner died (mutexes).

ESysEPERM = 1

System error: Operation not permitted.

ESysEPFNOSUPPORT = 96

System error: Protocol family not supported.

ESysEPIPE = 32

System error: Broken pipe.

ESysEPROTO = 71

System error: Protocol error.

ESysEPROTONOSUPPORT = 93

System error: Protocol not supported.

ESysEPROTOTYPE = 91

System error: Protocol wrong type for socket.

ESysERANGE = 34

System error: Math result not representable.

ESysEREMCHG = 78

System error: Remote address changed.

ESysEREMOTE = 66

System error: Object is remote.

ESysEREMOTEIO = 121

System error: Remote I/O error.

ESysERESTART = 85

System error: Interrupted system call should be restarted.

ESysERFKILL = 132

Operation not possible due to RF-Kill (wireless).

ESysEROFS = 30

System error: Read-only file system.

ESysESHUTDOWN = 108

System error: Cannot send after transport endpoint shutdown.

ESysESOCKTNOSUPPORT = 94

System error: Socket type not supported.

ESysESPIPE = 29

System error: Illegal seek.

ESysESRCH = 3

System error: No such process.

ESysESRMNT = 69

System error: Srmount error.

ESysESTALE = 116

System error: Stale NFS file handle.

ESysESTRPIPE = 86

System error: Streams pipe error.

ESysETIME = 62

System error: Timer expired.

ESysETIMEDOUT = 110

System error: Connection timed out.

ESysETOOMANYREFS = 109

System error: Too many references: cannot splice.

ESysETXTBSY = 26

System error: Text (code segment) file busy.

ESysEUCLEAN = 117

System error: Structure needs cleaning.

ESysEUNATCH = 49

System error: Protocol driver not attached.

ESysEUSERS = 87

System error: Too many users.

ESysEWOULDBLOCK = ESysEAGAIN

System error: Operation would block.

ESysEXDEV = 18

System error: Cross-device link.

ESysEXFULL = 54

System error: Exchange full.

FD_MAXFDSET = 1024

Maximum elements in a TFDSet (190) array.

FPE_FLTDIV = 3

Value signalling floating point divide by zero in case of SIGFPE signal.

FPE_FLTINV = 7

Value signalling floating point invalid operation in case of SIGFPE signal.

FPE_FLTOVF = 4

Value signalling floating point overflow in case of SIGFPE signal.

FPE_FLTRES = 6

Value signalling floating point inexact result in case of SIGFPE signal.

FPE_FLTSUB = 8

Value signalling floating point subscript out of range in case of SIGFPE signal.

FPE_FLTUND = 5

Value signalling floating point underflow in case of SIGFPE signal.

FPE_INTDIV = 1

Value signalling integer divide in case of SIGFPE signal.

FPE_INTOVF = 2

Value signalling integer overflow in case of SIGFPE signal.

F_GetFd = 1

fpFCntl (202) command: Get close-on-exec flag.

F_GetFl = 3

fpFCntl (202) command: Get file descriptor flags.

F_GetLk = 5

fpFCntl (202) command: Get lock.

F_GetOwn = 9

fpFCntl (202) command: get owner of file descriptor events.

F_OK = 0

`fpAccess` (193) call test: file exists.

`F_SetFd` = 2

`fpFCntl` (202) command: Set close-on-exec flag.

`F_SetFl` = 4

`fpFCntl` (202) command: Set file descriptor flags.

`F_SetLk` = 6

`fpFCntl` (202) command: Set lock.

`F_SetLkW` = 7

`fpFCntl` (202) command: Test lock.

`F_SetOwn` = 8

`fpFCntl` (202) command: Set owner of file descriptor events.

`ln2bitmask` = 1 shl `ln2bitsinword` - 1

Last bit in word.

`ln2bitsinword` = 6

Power of 2 number of bits in word.

`MAP_ANON` = `MAP_ANONYMOUS`

Anonymous memory mapping (data private to application).

`MAP_ANONYMOUS` = \$20

`FpMMap` (216) map type: Don't use a file.

`MAP_DENYWRITE` = \$800

`FpMMap` (216) option: Ignored.

`MAP_EXECUTABLE` = \$1000

`FpMMap` (216) option: Ignored.

`MAP_FAILED` = `pointer(- 1)`

Memory mapping failed error code.

`MAP_FIXED` = \$10

FpMMap (216) map type: Interpret addr exactly.

MAP_GROWSDOWN = \$100

FpMMap (216) option: Memory grows downward (like a stack).

MAP_LOCKED = \$2000

FpMMap (216) option: lock the pages in memory.

MAP_NORESERVE = \$4000

FpMMap (216) option: Do not reserve swap pages for this memory.

MAP_PRIVATE = \$2

FpMMap (216) map type: Changes are private.

MAP_SHARED = \$1

FpMMap (216) map type: Share changes.

MAP_TYPE = \$f

FpMMap (216) map type: Bitmask for type of mapping.

NAME_MAX = UnixType.NAME_MAX

Maximum filename length.

O_APPEND = \$400

fpOpen (220) file open mode: Append to file.

O_CREAT = \$40

fpOpen (220) file open mode: Create if file does not yet exist.

O_DIRECT = \$4000

fpOpen (220) file open mode: Minimize caching effects.

O_DIRECTORY = \$10000

fpOpen (220) file open mode: File must be directory.

O_EXCL = \$80

fpOpen (220) file open mode: Open exclusively.

O_LARGEFILE = \$8000

`fpOpen (220)` file open mode: Open for 64-bit I/O.

`O_NDELAY = O_NONBLOCK`

`fpOpen (220)` file open mode: Alias for `O_NonBlock (170)`.

`O_NOCTTY = $100`

`fpOpen (220)` file open mode: No TTY control.

`O_NOFOLLOW = $20000`

`fpOpen (220)` file open mode: Fail if file is symbolic link.

`O_NONBLOCK = $800`

`fpOpen (220)` file open mode: Open in non-blocking mode.

`O_RDONLY = 0`

`fpOpen (220)` file open mode: Read only.

`O_RDWR = 2`

`fpOpen (220)` file open mode: Read/Write.

`O_SYNC = $1000`

`fpOpen (220)` file open mode: Write to disc at once.

`O_TRUNC = $200`

`fpOpen (220)` file open mode: Truncate file to length 0.

`O_WRONLY = 1`

`fpOpen (220)` file open mode: Write only.

`PATH_MAX = UnixType.PATH_MAX`

Maximum pathname length.

`POLLERR = $0008`

Error condition on output file descriptor.

`POLLHUP = $0010`

Hang up.

`POLLIN = $0001`

Data is available for reading.

POLLNVAL = \$0020

Invalid request, file descriptor not open.

POLLOUT = \$0004

Writing data will not block the write call.

POLLPRI = \$0002

Urgent data is available for reading.

POLLRDBAND = \$0080

Priority data ready for reading.

POLLRDNORM = \$0040

Same as POLLIN.

POLLWRBAND = \$0200

Priority data may be written.

POLLWRNORM = \$0100

Equivalent to POLLOUT.

PRIO_PGRP = `UnixType.PRIO_PGRP`

Easy access alias for `unixtype.PRIO_PGRP` ([2230](#)).

PRIO_PROCESS = `UnixType.PRIO_PROCESS`

Easy access alias for `unixtype.PRIO_PROCESS` ([2230](#)).

PRIO_USER = `UnixType.PRIO_USER`

Easy access alias for `unixtype.PRIO_USER` ([2230](#)).

PROT_EXEC = \$4

`FpMMap` ([216](#)) memory access: page can be executed.

PROT_NONE = \$0

`FpMMap` ([216](#)) memory access: page can not be accessed.

PROT_READ = \$1

FpMMap (216) memory access: page can be read.

PROT_WRITE = 2

FpMMap (216) memory access: page can be written.

RLIMIT_AS = 9

RLimit request address space limit.

RLIMIT_CORE = 4

RLimit request max core file size.

RLIMIT_CPU = 0

RLimit request CPU time in ms.

RLIMIT_DATA = 2

RLimit request max data size.

RLIMIT_FSIZE = 1

Rlimit request maximum file size.

RLIMIT_LOCKS = 10

RLimit request maximum file locks held.

RLIMIT_MEMLOCK = 8

RLimit request max locked-in-memory address space.

RLIMIT_NOFILE = 7

RLimit request max number of open files.

RLIMIT_NPROC = 6

RLimit request max number of processes.

RLIMIT_RSS = 5

RLimit request max resident set size.

RLIMIT_STACK = 3

RLimit request max stack size.

R_OK = 4

fpAccess ([193](#)) call test: read allowed.

SA_INTERRUPT = \$20000000

Sigaction options: ?

SA_NOCLDSTOP = 1

Sigaction options: Do not receive notification when child processes stop.

SA_NOCLDWAIT = 2

Sigaction options: ?

SA_NODEFER = \$40000000

Sigaction options: Do not mask signal in its own signal handler.

SA_NOMASK = SA_NODEFER

Sigaction options: Do not prevent the signal from being received when it is handled.

SA_ONESHOT = SA_RESETHAND

Sigaction options: Restore the signal action to the default state.

SA_ONSTACK = \$08000000

SA_ONSTACK is used in the fpsigaction ([232](#)) to indicate the signal handler must be called on an alternate signal stack provided by sigaltstack(2) If an alternate stack is not available, the default stack will be used.

SA_RESETHAND = \$80000000

Sigaction options: Restore signal action to default state when signal handler exits.

SA_RESTART = \$10000000

Sigaction options: Provide behaviour compatible with BSD signal semantics.

SA_RESTORER = \$04000000

Signal restorer handler.

SA_SIGINFO = 4

Sigaction options: The signal handler takes 3 arguments, not one.

SEEK_CUR = 1

fpLSeek ([213](#)) option: Set position relative to current position.

SEEK_END = 2

fpLSeek (213) option: Set position relative to end of file.

SEEK_SET = 0

fpLSeek (213) option: Set absolute position.

SIGABRT = 6

Signal: ABRT (Abort).

SIGALRM = 14

Signal: ALRM (Alarm clock).

SIGBUS = 7

Signal: BUS (bus error).

SIGCHLD = 17

Signal: CHLD (child status changed).

SIGCONT = 18

Signal: CONT (Continue).

SIGFPE = 8

Signal: FPE (Floating point error).

SIGHUP = 1

Signal: HUP (Hangup).

SIGILL = 4

Signal: ILL (Illegal instruction).

SIGINT = 2

Signal: INT (Interrupt).

SIGIO = 29

Signal: IO (I/O operation possible).

SIGIOT = 6

Signal: IOT (IOT trap).

SIGKILL = 9

Signal: KILL (unblockable).

SIGPIPE = 13

Signal: PIPE (Broken pipe).

SIGPOLL = SIGIO

Signal: POLL (Pollable event).

SIGPROF = 27

Signal: PROF (Profiling alarm).

SIGPWR = 30

Signal: PWR (power failure restart).

SIGQUIT = 3

Signal: QUIT.

SIGSEGV = 11

Signal: SEGV (Segmentation violation).

SIGSTKFLT = 16

Signal: STKFLT (Stack Fault).

SIGSTOP = 19

Signal: STOP (Stop, unblockable).

SIGTERM = 15

Signal: TERM (Terminate).

SIGTRAP = 5

Signal: TRAP (Trace trap).

SIGTSTP = 20

Signal: TSTP (keyboard stop).

SIGTTIN = 21

Signal: TTIN (Terminal input, background).

SIGTTOU = 22

Signal: TTOU (Terminal output, background).

SIGUNUSED = 31

Signal: Unused.

SIGURG = 23

Signal: URG (Socket urgent condition).

SIGUSR1 = 10

Signal: USR1 (User-defined signal 1).

SIGUSR2 = 12

Signal: USR2 (User-defined signal 2).

SIGVTALRM = 26

Signal: VTALRM (Virtual alarm clock).

SIGWINCH = 28

Signal: WINCH (Window/Terminal size change).

SIGXCPU = 24

Signal: XCPU (CPU limit exceeded).

SIGXFSZ = 25

Signal: XFSZ (File size limit exceeded).

SIG_BLOCK = 0

Sigprocmask flags: Add signals to the set of blocked signals.

SIG_DFL = 0

Signal handler: Default signal handler.

SIG_ERR = - 1

Signal handler: error.

SIG_IGN = 1

Signal handler: Ignore signal.

`SIG_MAXSIG = UnixType.SIG_MAXSIG`

Maximum system signal number.

`SIG_SETMASK = 2`

Sigprocmask flags: Set of blocked signals is given.

`SIG_UNBLOCK = 1`

Sigprocmask flags: Remove signals from the set set of blocked signals.

`SI_PAD_SIZE = 128 div sizeof(longint) - 3`

Signal information pad size.

`SYS_NMLN = UnixType.SYS_NMLN`

Max system name length.

`S_IFBLK = 24576`

File (`#rtl.baseunix.stat (249)` record) mode: Block device.

`S_IFCHR = 8192`

File (`#rtl.baseunix.stat (249)` record) mode: Character device.

`S_IFDIR = 16384`

File (`#rtl.baseunix.stat (249)` record) mode: Directory.

`S_IFIFO = 4096`

File (`#rtl.baseunix.stat (249)` record) mode: FIFO.

`S_IFLNK = 40960`

File (`#rtl.baseunix.stat (249)` record) mode: Link.

`S_IFMT = 61440`

File (`#rtl.baseunix.stat (249)` record) mode: File type bit mask.

`S_IFREG = 32768`

File (`#rtl.baseunix.stat (249)` record) mode: Regular file.

`S_IFSOCK = 49152`

File (`#rtl.baseunix.stat (249)` record) mode: Socket.

S_IRGRP = %0000100000

Mode flag: Read by group.

S_IROTH = %00000000100

Mode flag: Read by others.

S_IRUSR = %0100000000

Mode flag: Read by owner.

S_IRWXG = S_IRGRP or S_IWGRP or S_IXGRP

Mode flag: Read, write, execute by groups.

S_IRWXO = S_IROTH or S_IWOTH or S_IXOTH

Mode flag: Read, write, execute by others.

S_IRWXU = S_IRUSR or S_IWUSR or S_IXUSR

Mode flag: Read, write, execute by user.

S_ISGID = &2000

Mode flag: Set Group ID on execution.

S_ISUID = &4000

Mode flag: Set user ID on execution.

S_ISVTX = &1000

Mode flag: Set sticky bit.

S_IWGRP = %0000010000

Mode flag: Write by group.

S_IWOTH = %0000000010

Mode flag: Write by others.

S_IWUSR = %0010000000

Mode flag: Write by owner.

S_IXGRP = %0000001000

Mode flag: Execute by group.

S_IXOTH = %0000000001

Mode flag: Execute by others.

S_IXUSR = %0001000000

Mode flag: Execute by owner.

UTSNAME_DOMAIN_LENGTH = UTSNAME_LENGTH

Max length of utsname (255) domain name.

UTSNAME_LENGTH = SYS_NMLN

Max length of utsname (255) system name, release, version, machine.

UTSNAME_NODENAME_LENGTH = UTSNAME_LENGTH

Max length of utsname (255) node name.

WNOHANG = 1

#rtl.baseunix.fpWaitpid (245) option: Do not wait for processes to terminate.

wordsinfdset = FD_MAXFDSET div BITSINWORD

Number of words in a TFDSet (190) array.

wordsinsigset = SIG_MAXSIG div BITSINWORD

Number of words in a signal set.

WUNTRACED = 2

#rtl.baseunix.fpWaitpid (245) option: Also report children which were stopped but not yet reported.

W_OK = 2

fpAccess (193) call test: write allowed.

X_OK = 1

fpAccess (193) call test: execute allowed.

_STAT_VER = _STAT_VER_LINUX

Stat version number.

_STAT_VER_KERNEL = 0

Current version of stat record.

_STAT_VER_LINUX = 1

Version of Linux stat record.

1.3.2 Types

`Blkcnt64_t = cuint64`

64-bit block count.

`Blkcnt_t = cuint`

Block count type.

`Blksize_t = cuint`

Block size type.

`cbool = UnixType.cbool`

Boolean type.

`cchar = UnixType.cchar`

Alias for `#rtl.UnixType.cchar` ([2231](#)).

`cdouble = UnixType.cdouble`

Double precision real format.

`cfloat = UnixType.cfloat`

Floating-point real format.

`cint = UnixType.cint`

C type: integer (natural size).

`cint16 = UnixType.cint16`

C type: 16 bits sized, signed integer.

`cint32 = UnixType.cint32`

C type: 32 bits sized, signed integer.

`cint64 = UnixType.cint64`

C type: 64 bits sized, signed integer.

`cint8 = UnixType.cint8`

C type: 8 bits sized, signed integer.

`clock_t = UnixType.clock_t`

Clock ticks type.

```
clong = UnixType.clong
```

C type: long signed integer (double sized).

```
clonglong = UnixType.clonglong
```

C type: 64-bit (double long) signed integer.

```
coff_t = UnixType.TOff
```

Character offset type.

```
cschar = UnixType.cschar
```

Signed character type.

```
cshort = UnixType.cshort
```

C type: short signed integer (half sized).

```
csigned = UnixType.csigned
```

`csigned` is an alias for `cint` ([180](#)).

```
csint = UnixType.csint
```

Signed integer.

```
csize_t = UnixType.size_t
```

Character size type.

```
cslong = UnixType.cslong
```

The size is CPU dependent.

```
cslonglong = UnixType.cslonglong
```

`cslonglong` is an alias for `clonglong` ([181](#)).

```
csshort = UnixType.csshort
```

Short signed integer type.

```
cuchar = UnixType.cuchar
```

Alias for `#rtl.UnixType.cuchar` ([2232](#)).

```
cuint = UnixType.cuint
```

C type: unsigned integer (natural size).

`cuint16 = UnixType.cuint16`

C type: 16 bits sized, unsigned integer.

`cuint32 = UnixType.cuint32`

C type: 32 bits sized, unsigned integer.

`cuint64 = UnixType.cuint64`

C type: 64 bits sized, unsigned integer.

`cuint8 = UnixType.cuint8`

C type: 8 bits sized, unsigned integer.

`culong = UnixType.culong`

C type: long unsigned integer (double sized).

`culonglong = UnixType.culonglong`

C type: 64-bit (double long) unsigned integer.

`cunsigned = UnixType.cunsigned`

Alias for `#rtl.unixtype.cunsigned` ([2233](#)).

`cushort = UnixType.cushort`

C type: short unsigned integer (half sized).

`dev_t = UnixType.dev_t`

Device descriptor type.

`gid_t = UnixType.gid_t`

Group ID type.

`ino_t = UnixType.ino_t`

Inode type.

`kernel_gid_t = cuint`

`kernel_gid_t` may differ from the `libc` type used to describe group IDs.

`kernel_loff_t = clonglong`

Long kernel offset type.

`kernel_mode_t = cuint`

`kernel_mode_t` may differ from the `libc` type used to describe file modes.

`kernel_off_t = clong`

Kernel offset type.

`kernel_uid_t = cuint`

`kernel_uid_t` may differ from the `libc` type used to describe user IDs.

`mode_t = UnixType.mode_t`

Inode mode type.

`nlink_t = UnixType.nlink_t`

Number of links type.

`off_t = UnixType.off_t`

Offset type.

`PBlkCnt = ^Blkcnt_t`

pointer to `TBlkCnt` (190) type.

`PBlkSize = ^Blksize_t`

Pointer to `TBlkSize` (190) type.

`pcbool = UnixType.pcbbool`

Pointer to boolean type `cbool` (180)

`pcchar = UnixType.pcchar`

Alias for `#rtl.UnixType.pcchar` (2234).

`pcdouble = UnixType.pcdouble`

Pointer to `cdouble` (180) type.

`pcfloat = UnixType.pcfloat`

Pointer to `cfloat` (180) type.

`pcint = UnixType.pcint`

Pointer to cInt (180) type.

```
pcint16 = UnixType.pcint16
```

Pointer to 16-bit signed integer type.

```
pcint32 = UnixType.pcint32
```

Pointer to signed 32-bit integer type.

```
pcint64 = UnixType.pcint64
```

Pointer to signed 64-bit integer type.

```
pcint8 = UnixType.pcint8
```

Pointer to 8-bits signed integer type.

```
pClock = UnixType.pClock
```

Pointer to TClock (190) type.

```
pclong = UnixType.pclong
```

Pointer to cLong (181) type.

```
pclonglong = UnixType.pclonglong
```

Pointer to longlong type.

```
pcschar = UnixType.pcschar
```

Pointer to character type cchar (181).

```
pcshort = UnixType.pcshort
```

Pointer to cShort (181) type.

```
pcsigned = UnixType.pcsigned
```

Pointer to signed integer type csigned (181).

```
pcsint = UnixType.pcsint
```

Pointer to signed integer type csint (181)

```
pcsize_t = UnixType.psize_t
```

Pointer to csize_t.

```
pcslong = UnixType.pcslong
```

Pointer to the signed long cslong ([181](#))

```
pcslonglong = UnixType.pcslonglong
```

Pointer to Signed longlong type cslonglong ([181](#))

```
pcssshort = UnixType.pcssshort
```

Pointer to short signed integer type csshort ([181](#))

```
pcuchar = UnixType.pcuchar
```

Alias for #rtl.UnixType.pcuchar ([2235](#)).

```
pcuint = UnixType.pcuint
```

Pointer to cUInt ([182](#)) type.

```
pcuint16 = UnixType.pcuint16
```

Pointer to 16-bit unsigned integer type.

```
pcuint32 = UnixType.pcuint32
```

Pointer to unsigned 32-bit integer type.

```
pcuint64 = UnixType.pcuint64
```

Pointer to unsigned 64-bit integer type.

```
pcuint8 = UnixType.pcuint8
```

Pointer to 8-bits unsigned integer type.

```
pculong = UnixType.pculong
```

Pointer to cuLong ([182](#)) type.

```
pculonglong = UnixType.pculonglong
```

Unsigned longlong type.

```
pcunsigned = UnixType.pcunsigned
```

Alias for #rtl.unixtype.pcunsigned ([2236](#)).

```
pcushort = UnixType.pcushort
```

Pointer to cuShort ([182](#)) type.

```
pDev = UnixType.pDev
```

Pointer to TDev (190) type.

```
pDir = ^Dir
```

Pointer to TDir (190) record.

```
pDirent = ^Dirent
```

Pointer to TDirent (190) record.

```
pFDSet = ^TFDSet
```

Pointer to TFDSet (190) type.

```
pFilDes = ^TFilDes
```

Pointer to TFilDes (190) type.

```
pfpstate = ^tfpstate
```

Pointer to tfpstate (250) record.

```
pGid = UnixType.pGid
```

Pointer to TGid (190) type.

```
pGrpArr = ^TGrpArr
```

Pointer to TGrpArr (190) array.

```
pid_t = UnixType.pid_t
```

Process ID type.

```
pIno = UnixType.pIno
```

Pointer to TIno (191) type.

```
piovec = ^tiovec
```

pointer to a iovec (248) record.

```
pMode = UnixType.pMode
```

Pointer to TMode (191) type.

```
pnLink = UnixType.pnLink
```

Pointer to TnLink (191) type.

```
pOff = UnixType.pOff
```

Pointer to TOff (191) type.

```
pPid = UnixType.pPid
```

Pointer to TPid (191) type.

```
ppollfd = ^pollfd
```

Pointer to tpollfd.

```
PRLimit = ^TRLimit
```

Pointer to TRLimit (252) record.

```
psigactionrec = ^sigactionrec
```

Pointer to SigActionRec (249) record type.

```
PSigContext = ^TSigContext
```

Pointer to #rtl.baseunix.TSigContext (253) record type.

```
psiginfo = ^tsiginfo
```

Pointer to #rtl.baseunix.TSigInfo (254) record type.

```
psigset = ^tsigset
```

Pointer to SigSet (189) type.

```
pSize = UnixType.pSize
```

Pointer to TSize (192) type.

```
pSize_t = UnixType.pSize_t
```

Pointer to Size_t.

```
pSocklen = UnixType.pSocklen
```

Pointer to TSockLen (192) type.

```
psSize = UnixType.psSize
```

Pointer to TsSize (192) type.

```
PStat = ^Stat
```

Pointer to TStat (192) type.

```
pstatfs = UnixType.PStatFs
```

This is an alias for the type defined in the `#rtl.unixtype` (2229) unit.

```
pthread_cond_t = UnixType.pthread_cond_t
```

Thread conditional variable type.

```
pthread_mutex_t = UnixType.pthread_mutex_t
```

Thread mutex type.

```
pthread_t = UnixType.pthread_t
```

POSIX thread type.

```
pTime = UnixType.pTime
```

Pointer to TTime (192) type.

```
ptimespec = UnixType.ptimespec
```

Pointer to timespec (190) type.

```
ptimeval = UnixType.ptimeval
```

Pointer to timeval (190) type.

```
ptimezone = ^timezone
```

Pointer to TimeZone (251) record.

```
ptime_t = UnixType.ptime_t
```

Pointer to time_t (191) type.

```
PTms = ^tms
```

Pointer to TTms (192) type.

```
Pucontext = ^Tucontext
```

Pointer to TUContext (254) type.

```
pUId = UnixType.pUId
```

Pointer to TUID (192) type.

```
pUtimBuf = ^UTimBuf
```

Pointer to TUTimBuf (192) type.

```
PUtsName = ^TUTsName
```

Pointer to TUnitsName (192) type.

```
rlim_t = culong
```

rlim_t is used as the type for the various fields in the TRLimit (252) record.

```
sigactionhandler = sigactionhandler_t
```

When installing a signal handler, the actual signal handler must be of type SigActionHandler.

```
sigactionhandler_t = procedure(signal: LongInt; info: psiginfo;
    context: PSigContext)
```

Standard signal action handler prototype.

```
signalhandler = signalhandler_t
```

Simple signal handler prototype.

```
signalhandler_t = procedure(signal: LongInt)
```

Standard signal handler prototype.

```
sigrestorerhandler = sigrestorerhandler_t
```

Alias for sigrestorerhandler_t (189) type.

```
sigrestorerhandler_t = procedure
```

Standard signal action restorer prototype.

```
sigset = sigset_t
```

Signal set type.

```
sigset_t = Array[0..wordsinsigset-1] of culong
```

Signal set type.

```
size_t = UnixType.size_t
```

Size specification type.

```
socklen_t = UnixType.socklen_t
```

Socket address length type.

```
ssize_t = UnixType.ssize_t
```

Small size type.

TBlkCnt = Blkcnt_t

Alias for Blkcnt_t (180) type.

TBlkSize = Blksize_t

Alias for blksize_t (180) type.

TClock = UnixType.TClock

Alias for clock_t (181) type.

TDev = UnixType.TDev

Alias for dev_t (182) type.

TDIr = Dir

Alias for Dir (247) type.

TDirent = Dirent

Alias for Dirent (248) type.

TFDSet = Array[0..(FD_MAXFDSETdivBITSINWORD)-1] of TFDSetEl

File descriptor set for fpSelect (228) call.

TFDSetEl = culong

Type alias for an element of the TFDSet.

TFilDes = Array[0..1] of cint

Array of file descriptors as used in fpPipe (222) call.

TGid = UnixType.TGid

Alias for gid_t (182) type.

TGrpArr = Array[0..0] of TGid

Array of gid_t (182) IDs.

timespec = UnixType.timespec

Short time specification type.

timeval = UnixType.timeval

Time specification type.

`time_t = UnixType.time_t`

Time span type.

`TIno = UnixType.TIno`

Alias for `ino_t` (182) type.

`TIOCtlRequest = UnixType.TIOCtlRequest`

Easy access alias for `unixtype.TIOCtlRequest` (2239).

`tiovec = iovec`

Alias for the `iovec` (248) record type.

`TMode = UnixType.TMode`

Alias for `mode_t` (183) type.

`TnLink = UnixType.TnLink`

Alias for `nlink_t` (183) type.

`TOff = UnixType.TOff`

Alias for `off_t` (183) type.

`TPid = UnixType.TPid`

Alias for `pid_t` (186) type.

`tpollfd = pollfd`

Alias for `pollfd` type.

`tsigactionhandler = sigactionhandler_t`

Alias for `sigactionhandler_t` (189) type.

`tsignalhandler = signalhandler_t`

Alias for `signalhandler_t` (189) type.

`tsigrestorerhandler = sigrestorerhandler_t`

Alias for `sigrestorerhandler_t` (189) type.

`tsigset = sigset_t`

Alias for `SigSet` (189) type.

`TSize = UnixType.TSize`

Alias for `size_t` (189) type.

`TSocklen = UnixType.TSocklen`

Alias for `socklen_t` (189) type.

`TsSize = UnixType.TsSize`

Alias for `ssize_t` (189) type.

`TStat = Stat`

Alias for `Stat` (249) type.

`tstatfs = UnixType.TStatFs`

Record describing a file system in the `unix.fstatfs` (155) call.

`TTime = UnixType.TTime`

Alias for `TTime` (192) type.

`Ttimespec = UnixType.Ttimespec`

Alias for `TimeSpec` (190) type.

`TTimeVal = UnixType.TTimeVal`

Alias for `timeval` (190) type.

`TTimeZone = timezone`

Alias for `TimeZone` (251) record.

`TTms = tms`

Alias for `Tms` (251) record type.

`TUid = UnixType.TUid`

Alias for `uid_t` (192) type.

`TUtimBuf = UtimBuf`

Alias for `UtimBuf` (254) type.

`TUtsName = UtsName`

Alias for `UtsName` (255) type.

`uid_t = UnixType.uid_t`

User ID type.

1.4 Procedures and functions

1.4.1 CreateShellArgV

Synopsis: Create a null-terminated array of strings from a command-line string.

Declaration: `function CreateShellArgV(const prog: string) : PPChar`
`function CreateShellArgV(const prog: RawByteString) : PPChar`

Visibility: default

Description: `CreateShellArgV` creates a command-line string for executing a shell command using 'sh -c'. The result is a null-terminated array of null-terminated strings suitable for use in `fpExecv` (200) and friends.

Errors: If no more memory is available, a heap error may occur.

See also: `fpExecv` (200), `FreeShellArgV` (246)

1.4.2 FpAccess

Synopsis: Check file access.

Declaration: `function FpAccess(pathname: PChar; aMode: cint) : cint`
`function FpAccess(const pathname: RawByteString; aMode: cint) : cint`

Visibility: default

Description: `FpAccess` tests user's access rights on the specified file. Mode is a mask existing of one or more of the following:

R_OK User has read rights.

W_OK User has write rights.

X_OK User has execute rights.

F_OK File exists.

The test is done with the real user ID, instead of the effective user ID. If the user has the requested rights, zero is returned. If access is denied, or an error occurred, a nonzero value is returned.

Errors: Extended error information can be retrieved using `fpGetErrno` (207).

sys_eaccess The requested access is denied, either to the file or one of the directories in its path.

sys_einval Mode was incorrect.

sys_enoent A directory component in `Path` doesn't exist or is a dangling symbolic link.

sys_enotdir A directory component in `Path` is not a directory.

sys_enomem Insufficient kernel memory.

sys_eloop `Path` has a circular symbolic link.

See also: `FpChown` (196), `FpChmod` (195)

Listing: `./examples/bunixex/ex26.pp`

Program Example26;

{ Program to demonstrate the Access function. }

Uses BaseUnix;

```
begin
  if fpAccess ('/etc/passwd',W_OK)=0 then
    begin
      Writeln ('Better check your system.');
```

```
      Writeln ('I can write to the /etc/passwd file !');
    end;
```

```
end.
```

1.4.3 FpAlarm

Synopsis: Schedule an alarm signal to be delivered.

Declaration: `function FpAlarm(seconds: cuint) : cuint`

Visibility: default

Description: `FpAlarm` schedules an alarm signal to be delivered to your process in `Seconds` seconds. When `Seconds` seconds have elapsed, the system will send a `SIGALRM` signal to the current process. If `Seconds` is zero, then no new alarm will be set. Whatever the value of `Seconds`, any previous alarm is cancelled.

The function returns the number of seconds till the previously scheduled alarm was due to be delivered, or zero if there was none. A negative value indicates an error.

See also: [fpSigAction \(232\)](#), [fpPause \(222\)](#)

Listing: `./examples/bunixex/ex59.pp`

Program Example59;

{ Program to demonstrate the Alarm function. }

Uses BaseUnix;

Procedure `AlarmHandler(Sig : cint);cdecl;`

```
begin
  Writeln ('Got to alarm handler');
```

```
end;
```

```
begin
  Writeln('Setting alarm handler');
  fpSignal(SIGALRM,SignalHandler(@AlarmHandler));
  Writeln ('Scheduling Alarm in 10 seconds');
  fpAlarm(10);
  Writeln ('Pausing');
  fpPause;
  Writeln ('Pause returned');
```

```
end.
```

1.4.4 FpChdir

Synopsis: Change current working directory.

Declaration: `function FpChdir(path: PChar) : cint`
`function FpChdir(const path: RawByteString) : cint`

Visibility: default

Description: `fpChDir` sets the current working directory to `Path`.

It returns zero if the call was successful, -1 on error.

Note: There exist a portable alternative to `fpChDir`: `system.chdir`. Please use `fpChDir` only if you are writing Unix specific code. `System.chdir` will work on all operating systems.

Errors: Extended error information can be retrieved using `fpGetErrno` (207).

See also: `fpGetCwd` (206)

1.4.5 FpChmod

Synopsis: Change file permission bits.

Declaration: `function FpChmod(path: PChar; Mode: TMode) : cint`
`function FpChmod(const path: RawByteString; Mode: TMode) : cint`

Visibility: default

Description: `fpChmod` sets the `Mode` bits of the file in `Path` to `Mode`. `Mode` can be specified by 'or'-ing the following values:

S_ISUID Set user ID on execution.

S_ISGID Set Group ID on execution.

S_ISVTX Set sticky bit.

S_IRUSR Read by owner.

S_IWUSR Write by owner.

S_IXUSR Execute by owner.

S_IRGRP Read by group.

S_IWGRP Write by group.

S_IXGRP Execute by group.

S_IROTH Read by others.

S_IWOTH Write by others.

S_IXOTH Execute by others.

S_IRWXO Read, write, execute by others.

S_IRWXG Read, write, execute by groups.

S_IRWXU Read, write, execute by user.

If the function is successful, zero is returned. A nonzero return value indicates an error.

Errors: The following error codes are returned:

sys_eperm The effective UID doesn't match the ownership of the file, and is not zero. Owner or group were not specified correctly.

sys_eaccess One of the directories in `Path` has no search (=execute) permission.

sys_enoent A directory entry in `Path` does not exist or is a symbolic link pointing to a non-existent directory.

sys_enomem Insufficient kernel memory.

sys_erofs The file is on a read-only file system.

sys_eloop `Path` has a reference to a circular symbolic link, i.e. a symbolic link, whose expansion points to itself.

See also: `fpChown` (196), `fpAccess` (193)

Listing: `./examples/bunixex/ex23.pp`

Program `Example23;`

{ Program to demonstrate the Chmod function. }

Uses `BaseUnix, Unix;`

Var `F : Text;`

begin

```
{ Create a file }
Assign (f,'testex21');
Rewrite (F);
Writeln (f,'#!/bin/sh');
Writeln (f,'echo Some text for this file');
Close (F);
fpChmod ('testex21',&777);
{ File is now executable }
fpexecl ('./testex21',[]);
```

end.

1.4.6 FpChown

Synopsis: Change owner of file.

Declaration: `function FpChown(path: PChar; owner: TUid; group: TGid) : cint`
`function FpChown(const path: RawByteString; owner: TUid; group: TGid)`
`: cint`

Visibility: default

Description: `fpChown` sets the User ID and Group ID of the file in `Path` to `Owner, Group`.

The function returns zero if the call was successful, a nonzero return value indicates an error.

Errors: The following error codes are returned:

sys_eperm The effective UID doesn't match the ownership of the file, and is not zero. Owner or group were not specified correctly.

sys_eaccess One of the directories in `Path` has no search (=execute) permission.

sys_enoent A directory entry in `Path` does not exist or is a symbolic link pointing to a non-existent directory.

sys_enomem Insufficient kernel memory.

sys_erofs The file is on a read-only file system.

sys_eloop Path has a reference to a circular symbolic link, i.e. a symbolic link, whose expansion points to itself.

See also: `fpChmod` ([195](#)), `fpAccess` ([193](#))

Listing: `./examples/bunixex/ex24.pp`

Program Example24;

{ Program to demonstrate the Chown function. }

Uses BaseUnix;

Var UID : TUid;
 GID : TGid;
 F : Text;

begin

```

Writeln ('This will only work if you are root. ');
Write ('Enter a UID : '); readln(UID);
Write ('Enter a GID : '); readln(GID);
Assign (f, 'test.txt');
Rewrite (f);
Writeln (f, 'The owner of this file should become : ');
Writeln (f, 'UID : ', UID);
Writeln (f, 'GID : ', GID);
Close (F);
if fpChown ('test.txt', UID, GID) <> 0 then
  if fpgeterrno = ESysEPerm then
    Writeln ('You are not root !')
  else
    Writeln ('Chmod failed with exit code : ', fpgeterrno)
  else
    Writeln ('Changed owner successfully !');
end.
```

1.4.7 FpClose

Synopsis: Close file descriptor.

Declaration: `function FpClose(fd: cint) : cint`

Visibility: default

Description: `FpClose` closes a file with file descriptor `Fd`. The function returns zero if the file was closed successfully, a nonzero return value indicates an error.

For an example, see `FpOpen` ([220](#)).

Errors: Extended error information can be retrieved using `fpGetErrno` ([207](#)).

See also: `FpOpen` ([220](#)), `FpRead` ([224](#)), `FpWrite` ([245](#)), `FpFTruncate` ([205](#)), `FpLSeek` ([213](#))

1.4.8 FpClosedir

Synopsis: Close directory file descriptor.

Declaration: `function FpClosedir(var dirp: Dir) : cint`

Visibility: default

Description: `FpCloseDir` closes the directory pointed to by `dirp`. It returns zero if the directory was closed successfully, -1 otherwise.

For an example, see `fpOpenDir` (221).

Errors: Extended error information can be retrieved using `fpGetErrno` (207).

See also: `FpOpenDir` (221), `FpReadDir` (225)

1.4.9 FpDup

Synopsis: Duplicate a file handle.

Declaration: `function FpDup(fildes: cint) : cint`
`function FpDup(var oldfile: text; var newfile: text) : cint`
`function FpDup(var oldfile: File; var newfile: File) : cint`

Visibility: default

Description: `FpDup` returns a file descriptor that is a duplicate of the file descriptor `fildes`.

The second and third forms make `NewFile` an exact copy of `OldFile`, after having flushed the buffer of `OldFile` in case it is a Text file or untyped file. Due to the buffering mechanism of Pascal, these calls do not have the same functionality as the `dup` call in C. The internal Pascal buffers are not the same after this call, but when the buffers are flushed (e.g. after output), the output is sent to the same file. Doing an `lseek` will, however, work as in C, i.e. doing a `lseek` will change the file position in both files.

The function returns a negative value in case of an error, a positive value is a file handle, and indicates success.

Errors: A negative value can be one of the following error codes:

sys_ebadf `OldFile` hasn't been assigned.

sys_emfile Maximum number of open files for the process is reached.

See also: `fpDup2` (199)

Listing: `./examples/bunixex/ex31.pp`

```
program Example31;

{ Program to demonstrate the Dup function. }

uses baseunix;

var f : text;

begin
  if fpdup (output,f)=-1 then
    Writeln ('Dup Failed !');
  writeln ('This is written to stdout.');
```

```

writeln (f,'This is written to the dup file, and flushed');flush(f);
writeln
end.

```

1.4.10 FpDup2

Synopsis: Duplicate one file handle to another.

Declaration: `function FpDup2(fildes: cint; fildes2: cint) : cint`
`function FpDup2(var oldfile: text; var newfile: text) : cint`
`function FpDup2(var oldfile: File; var newfile: File) : cint`

Visibility: default

Description: Makes `fildes2` or `NewFile` an exact copy of `fildes` or `OldFile`, after having flushed the buffer of `OldFile` in the case of text or untyped files.

After a call to `fdup2`, the 2 file descriptors point to the same physical device (a file, socket, or a terminal).

`NewFile` can be an assigned file. If `newfile` or `fildes` was open, it is closed first. Due to the buffering mechanism of Pascal, this has not the same functionality as the `dup2` call in C. The internal Pascal buffers are not the same after this call, but when the buffers are flushed (e.g. after output), the output is sent to the same file. Doing an `lseek` will, however, work as in C, i.e. doing a `lseek` will change the file position in both files.

The function returns the new file descriptor number, on error -1 is returned, and the error can be retrieved with `fpgeterrno` (207)

Errors: In case of error, the following error codes can be reported:

sys_ebadf `OldFile` (or `fildes`) hasn't been assigned.

sys_emfile Maximum number of open files for the process is reached.

See also: `fpDup` (198)

Listing: `./examples/bunixex/ex32.pp`

```

program Example32;

{ Program to demonstrate the FpDup2 function. }

uses BaseUnix;

var f : text;
    i : longint;

begin
  Assign (f,'text.txt');
  Rewrite (F);
  For i:=1 to 10 do writeln (F,'Line : ',i);
  if fpdup2 (output,f)=-1 then
    Writeln ('Dup2 Failed !');
  writeln ('This is written to stdout. ');
  writeln (f,'This is written to the dup file, and flushed');
  flush(f);
  writeln;
  { Remove file. Comment this if you want to check flushing.}

```



```
    fpUnlink ('text.txt');
end.
```

1.4.11 FpExecv

Synopsis: Execute process.

Declaration: `function FpExecv(path: PChar; argv: PPChar) : cint`
`function FpExecv(const path: RawByteString; argv: PPChar) : cint`

Visibility: default

Description: Replaces the currently running program with the program, specified in `path`. It gives the program the options in `argv`. This is a pointer to an array of pointers to null-terminated strings. The last pointer in this array should be nil. The current environment is passed to the program. On success, `execv` does not return.

Errors: On error, -1 is returned. Extended error information can be retrieved with `fpGetErrNo` ([207](#))

sys_eaccess File is not a regular file, or has no execute permission. A component of the path has no search permission.

sys_eperm The file system is mounted .

sys_e2big Argument list too big.

sys_enoexec The magic number in the file is incorrect.

sys_enoent The file does not exist.

sys_enomem Not enough memory for kernel.

sys_enotdir A component of the path is not a directory.

sys_eloop The path contains a circular reference (via symlinks).

See also: `fpExecve` ([201](#)), `fpFork` ([204](#))

Listing: `./examples/bunixex/ex8.pp`

Program Example8;

{ Program to demonstrate the Execv function. }

Uses Unix, strings;

Const Arg0 : PChar = '/bin/lS';
 Arg1 : Pchar = '-l';

Var PP : PPchar;

```
begin
    GetMem (PP,3*SizeOf(Pchar));
    PP[0]:=Arg0;
    PP[1]:=Arg1;
    PP[3]:=Nil;
    { Execute '/bin/lS -l', with current environment }
    fpExecv ('/bin/lS',pp);
end.
```

1.4.12 FpExecve

Synopsis: Execute process using environment.

Declaration: `function FpExecve(path: PChar; argv: PPChar; envp: PPChar) : cint`
`function FpExecve(const path: RawByteString; argv: PPChar; envp: PPChar)`
`: cint`

Visibility: default

Description: Replaces the currently running program with the program, specified in `path`. It gives the program the options in `argv`, and the environment in `envp`. They are pointers to an array of pointers to null-terminated strings. The last pointer in this array should be `nil`. On success, `execve` does not return.

Errors: Extended error information can be retrieved with `fpGetErrno` (207), and includes the following:

sys_eaccess File is not a regular file, or has no execute permission. A component of the path has no search permission.

sys_eperm The file system is mounted .

sys_e2big Argument list too big.

sys_enoexec The magic number in the file is incorrect.

sys_enoent The file does not exist.

sys_enomem Not enough memory for kernel.

sys_enotdir A component of the path is not a directory.

sys_eloop The path contains a circular reference (via symlinks).

See also: `fpExecv` (200), `fpFork` (204)

Listing: `./examples/bunixex/ex7.pp`

Program Example7;

{ Program to demonstrate the Execve function. }

Uses BaseUnix, strings;

Const Arg0 : PChar = '/bin/l\$';
Arg1 : Pchar = '-l';

Var PP : PPchar;

begin

GetMem (PP,3*SizeOf(Pchar));
PP[0]:=Arg0;
PP[1]:=Arg1;
PP[3]:=Nil;
{ Execute '/bin/l\$ -l', with current environment }
{ Env\$ is defined in system.inc }
fpExecVe ('/bin/l\$',pp,envp);

end.

1.4.13 FpExit

Synopsis: Exit the current process.

Declaration: `procedure FpExit (Status: cint)`

Visibility: default

Description: `FpExit` exits the currently running process, and report `Status` as the exit status.

Remark If this call is executed, the normal unit finalization code will not be executed. This may lead to unexpected errors and stray files on your system. It is therefore recommended to use the `Halt` call instead.

Errors: None.

See also: `FpFork` (204), `FpExecve` (201)

1.4.14 FpFcntl

Synopsis: File control operations.

Declaration: `function FpFcntl(fildes: cint; cmd: cint) : cint`
`function FpFcntl(fildes: cint; cmd: cint; arg: cint) : cint`
`function FpFcntl(fildes: cint; cmd: cint; var arg: FLock) : cint`

Visibility: default

Description: Read/set a file's attributes. `Fildes` a valid file descriptor. `Cmd` specifies what to do, and is one of the following:

F_GetFd Read the `close_on_exec` flag. If the low-order bit is 0, then the file will remain open across `execve` calls.

F_GetFl Read the descriptor's flags.

F_GetOwn Get the Process ID of the owner of a socket.

F_SetFd Set the `close_on_exec` flag of `fildes`. (only the least significant bit is used).

F_GetLk Return the `flock` record that prevents this process from obtaining the lock, or set the `l_type` field of the lock of there is no obstruction. `Arg` is the `flock` record.

F_SetLk Set the lock or clear it (depending on `l_type` in the `flock` structure). if the lock is held by another process, an error occurs.

F_GetLkw Same as for **F_Setlk**, but wait until the lock is released.

F_SetOwn Set the Process or process group that owns a socket.

The function returns 0 if successful, -1 otherwise.

Errors: On error, -1 is returned. Use `fpGetErrno` (207) for extended error information.

sys_ebadf `Fd` has a bad file descriptor.

sys_eagain or sys_eaccess For , if the lock is held by another process.

1.4.15 **fpfdfillset**

Synopsis: Set all file descriptors in the set.

Declaration: `function fpfdfillset(var nset: TFDSet) : cint`

Visibility: default

Description: `fpfdfillset` sets all file descriptors in `nset`.

See also: [FpSelect \(228\)](#), [FpFD_ZERO \(204\)](#), [FpFD_IsSet \(203\)](#), [FpFD_Clr \(203\)](#), [FpFD_Set \(203\)](#)

1.4.16 **fpFD_CLR**

Synopsis: Clears a file descriptor in a set.

Declaration: `function fpFD_CLR(fdno: cint; var nset: TFDSet) : cint`

Visibility: default

Description: `FpFD_Clr` clears file descriptor `fdno` in file descriptor set `nset`.

For an example, see [FpSelect \(228\)](#).

Errors: None.

See also: [FpSelect \(228\)](#), [FpFD_ZERO \(204\)](#), [FpFD_Set \(203\)](#), [FpFD_IsSet \(203\)](#)

1.4.17 **fpFD_ISSET**

Synopsis: Check whether a file descriptor is set.

Declaration: `function fpFD_ISSET(fdno: cint; const nset: TFDSet) : cint`

Visibility: default

Description: `FpFD_Set` Checks whether file descriptor `fdNo` in file descriptor set `fds` is set. It returns zero if the descriptor is not set, 1 if it is set. If the number of the file descriptor is wrong, -1 is returned.

For an example, see [FpSelect \(228\)](#).

Errors: If an invalid file descriptor number is passed, -1 is returned.

See also: [FpSelect \(228\)](#), [FpFD_ZERO \(204\)](#), [FpFD_Clr \(203\)](#), [FpFD_Set \(203\)](#)

1.4.18 **fpFD_SET**

Synopsis: Set a file descriptor in a set.

Declaration: `function fpFD_SET(fdno: cint; var nset: TFDSet) : cint`

Visibility: default

Description: `FpFD_Set` sets file descriptor `fdno` in file descriptor set `nset`.

For an example, see [FpSelect \(228\)](#).

Errors: None.

See also: [FpSelect \(228\)](#), [FpFD_ZERO \(204\)](#), [FpFD_Clr \(203\)](#), [FpFD_IsSet \(203\)](#)

1.4.19 fpFD_ZERO

Synopsis: Clear all file descriptors in set.

Declaration: `function fpFD_ZERO(out nset: TFDSet) : cint`

Visibility: default

Description: `FpFD_ZERO` clears all the file descriptors in the file descriptor set `nset`.

For an example, see `FpSelect` (228).

Errors: None.

See also: `FpSelect` (228), `FpFD_Clr` (203), `FpFD_Set` (203), `FpFD_IsSet` (203)

1.4.20 FpFork

Synopsis: Create child process.

Declaration: `function FpFork : TPid`

Visibility: default

Description: `FpFork` creates a child process which is a copy of the parent process. `FpFork` returns the process ID in the parent process, and zero in the child's process. (you can get the parent's PID with `fpGetPPid` (209)).

Errors: On error, -1 is returned to the parent, and no child is created.

sys_eagain Not enough memory to create child process.

See also: `fpExecve` (201), `#rtl.linux.Clone` (1015)

1.4.21 FPFStat

Synopsis: Retrieve file information about a file descriptor.

Declaration: `function FpFStat(fd: cint; var sb: Stat) : cint`
`function FPFStat(var F: Text; var Info: Stat) : Boolean`
`function FPFStat(var F: File; var Info: Stat) : Boolean`

Visibility: default

Description: `FpFStat` gets information about the file specified in one of the following:

Fd a valid file descriptor.

F an opened text file or untyped file.

and stores it in `Info`, which is of type `stat` (249). The function returns zero if the call was successful, a nonzero return value indicates failure.

Errors: Extended error information can be retrieved using `fpGetErrno` (207).

sys_enoent Path does not exist.

See also: `FpStat` (237), `FpLStat` (214)

Listing: `./examples/bunixex/ex28.pp`

```

program example28;

{ Program to demonstrate the FStat function. }

uses BaseUnix;

var f : text;
    i : byte;
    info : stat;

begin
    { Make a file }
    assign (f,'test.fil');
    rewrite (f);
    for i:=1 to 10 do writeln (f,'Testline # ',i);
    close (f);
    { Do the call on made file. }
    if fpstat ('test.fil',info)<>0 then
        begin
            writeln('Fstat failed. Errno : ',fpgeterrno);
            halt (1);
        end;
    writeln;
    writeln ('Result of fstat on file "test.fil".');
    writeln ('Inode : ',info.st_ino);
    writeln ('Mode : ',info.st_mode);
    writeln ('nlink : ',info.st_nlink);
    writeln ('uid : ',info.st_uid);
    writeln ('gid : ',info.st_gid);
    writeln ('rdev : ',info.st_rdev);
    writeln ('Size : ',info.st_size);
    writeln ('Blksize : ',info.st_blksize);
    writeln ('Blocks : ',info.st_blocks);
    writeln ('atime : ',info.st_atime);
    writeln ('mtime : ',info.st_mtime);
    writeln ('ctime : ',info.st_ctime);
    { Remove file }
    erase (f);
end.

```

1.4.22 FpFtruncate

Synopsis: Truncate file on certain size.

Declaration: `function FpFtruncate(fd: cint; flength: TOff) : cint`

Visibility: default

Description: `FpFtruncate` sets the length of a file in `fd` on `flength` bytes, where `flength` must be less than or equal to the current length of the file in `fd`.

The function returns zero if the call was successful, a nonzero return value indicates that an error occurred.

Errors: Extended error information can be retrieved using `fpGetErrno` ([207](#)).

See also: `FpOpen` ([220](#)), `FpClose` ([197](#)), `FpRead` ([224](#)), `FpWrite` ([245](#)), `FpLSeek` ([213](#))

1.4.23 FpGetcwd

Synopsis: Retrieve the current working directory.

Declaration: `function FpGetcwd(path: PChar; siz: TSize) : PChar`
`function FpGetcwd : RawByteString`

Visibility: default

Description: `fpgetCWD` returns the current working directory of the running process. It is returned in `Path`, which points to a memory location of at least `siz` bytes.

If the function is successful, a pointer to `Path` is returned, or a string with the result. On error `Nil` or an empty string are returned.

Errors: On error `Nil` or an empty string are returned.

See also: [FpGetPID \(209\)](#), [FpGetUID \(211\)](#)

1.4.24 FpGetegid

Synopsis: Return effective group ID.

Declaration: `function FpGetegid : TGid`

Visibility: default

Description: `FpGetegid` returns the effective group ID of the currently running process.

Errors: None.

See also: [FpGetGid \(208\)](#), [FpGetUid \(211\)](#), [FpGetEUid \(207\)](#), [FpGetPid \(209\)](#), [FpGetPPid \(209\)](#), [fpSetUID \(231\)](#), [FpSetGid \(230\)](#)

Listing: `./examples/bunixex/ex18.pp`

Program Example18;

{ Program to demonstrate the GetGid and GetEGid functions. }

Uses BaseUnix;

begin

writeln ('Group Id = ',fpgetgid,' Effective group Id = ',fpgetegid);

end.

1.4.25 FpGetEnv

Synopsis: Return value of environment variable.

Declaration: `function FpGetEnv(name: PChar) : PChar`
`function FpGetEnv(name: string) : PChar`

Visibility: default

Description: `FPGetEnv` returns the value of the environment variable in `Name`. If the variable is not defined, `nil` is returned. The value of the environment variable may be the empty string. A `PChar` is returned to accommodate for strings longer than 255 bytes, `TERMCAP` and `LS_COLORS`, for instance.

Errors: None.

Listing: ./examples/bunixex/ex41.pp

Program Example41;

{ Program to demonstrate the GetEnv function. }

Uses BaseUnix;

begin

Writeln ('Path is : ',fpGetenv('PATH'));

end.

1.4.26 fpgeterrno

Synopsis: Retrieve extended error information.

Declaration: `function fpgeterrno : LongInt`

Visibility: default

Description: `fpgeterrno` returns extended information on the latest error. It is set by all functions that communicate with the kernel or C library.

Errors: None.

See also: `fpseterrno` ([229](#))

1.4.27 FpGeteuid

Synopsis: Return effective user ID.

Declaration: `function FpGeteuid : TUid`

Visibility: default

Description: `FpGeteuid` returns the effective user ID of the currently running process.

Errors: None.

See also: `FpGetUid` ([211](#)), `FpGetGid` ([208](#)), `FpGetEGid` ([206](#)), `FpGetPid` ([209](#)), `FpGetPPid` ([209](#)), `fpSetUID` ([231](#)), `FpSetGid` ([230](#))

Listing: ./examples/bunixex/ex17.pp

Program Example17;

{ Program to demonstrate the GetUid and GetEUid functions. }

Uses BaseUnix;

begin

writeln ('User Id = ',fpgetuid,' Effective user Id = ',fpgeteuid);

end.

1.4.28 FpGetgid

Synopsis: Return real group ID.

Declaration: `function FpGetgid : TGid`

Visibility: default

Description: `FpGetgid` returns the real group ID of the currently running process.

Errors: None.

See also: `FpGetEGid` (206), `FpGetUid` (211), `FpGetEUid` (207), `FpGetPid` (209), `FpGetPPid` (209), `fpSetUID` (231), `FpSetGid` (230)

Listing: `./examples/bunixex/ex18.pp`

Program Example18;

{ Program to demonstrate the GetGid and GetEGid functions. }

Uses BaseUnix;

begin

writeln ('Group Id = ',fpgetgid,' Effective group Id = ',fpgetegid);

end.

1.4.29 FpGetgroups

Synopsis: Get the list of supplementary groups.

Declaration: `function FpGetgroups(gidsetsize: cint; var grouplist: TGrpArr) : cint`

Visibility: default

Description: `FpGetgroups` returns up to `gidsetsize` groups in `GroupList`

If the function is successful, then number of groups that were stored is returned. On error, -1 is returned.

Errors: On error, -1 is returned. Extended error information can be retrieved with `fpGetErrNo` (207)

See also: `FpGetpgrp` (208), `FpGetGID` (208), `FpGetEGID` (206)

1.4.30 FpGetpgrp

Synopsis: Get process group ID.

Declaration: `function FpGetpgrp : TPid`

Visibility: default

Description: `FpGetpgrp` returns the process group ID of the current process.

Errors: None.

See also: `fpGetPID` (209), `fpGetPPID` (209), `FpGetGID` (208), `FpGetUID` (211)

1.4.31 FpGetpid

Synopsis: Return current process ID.

Declaration: `function FpGetpid : TPid`

Visibility: default

Description: `FpGetpid` returns the process ID of the currently running process.

Note: There exist a portable alternative to `fpGetpid`: `system.GetProcessID`. Please use `fpGetpid` only if you are writing Unix specific code. `System.GetProcessID` will work on all operating systems.

Errors: None.

See also: `FpGetPPid` ([209](#))

Listing: `./examples/bunixex/ex16.pp`

Program Example16;

{ Program to demonstrate the GetPid, GetPPid function. }

Uses BaseUnix;

begin

Writeln ('Process Id = ',fpgetpid,' Parent process Id = ',fpgetppid);

end.

1.4.32 FpGetppid

Synopsis: Return parent process ID.

Declaration: `function FpGetppid : TPid`

Visibility: default

Description: `FpGetppid` returns the Process ID of the parent process.

Errors: None.

See also: `FpGetPid` ([209](#))

Listing: `./examples/bunixex/ex16.pp`

Program Example16;

{ Program to demonstrate the GetPid, GetPPid function. }

Uses BaseUnix;

begin

Writeln ('Process Id = ',fpgetpid,' Parent process Id = ',fpgetppid);

end.

1.4.33 fpGetPriority

Synopsis: Return process priority.

Declaration: `function fpGetPriority(Which: cint; Who: cint) : cint`

Visibility: default

Description: `GetPriority` returns the priority with which a process is running. Which process(es) is determined by the `Which` and `Who` variables. Which can be one of the predefined `Prio_Process`, `Prio_PGrp`, `Prio_User`, in which case `Who` is the process ID, Process group ID or User ID, respectively.

For an example, see `FpNice` (219).

Errors: Error information is returned solely by the `FpGetErrno` (207) function: a priority can be a positive or negative value.

sys_esrch No process found using `which` and `who`.

sys_einval Which was not one of `Prio_Process`, `Prio_Grp` or `Prio_User`.

See also: `FpSetPriority` (230), `FpNice` (219)

1.4.34 FpGetRLimit

Synopsis: Get process resource limits.

Declaration: `function FpGetRLimit(resource: cint; rlim: PRLimit) : cint`

Visibility: default

Description: `FpGetRLimit` gets the resource limits for the current process: `resource` determines the resource of which the kernel should return the limits (one of the many `RLIMIT_*` constants). `rlim` should point to a `TRLimit` (252) record and on success will contain the resource limits.

The function returns zero if the resource limits were correctly returned.

Errors: On error, -1 is returned and `fpgeterrno` (207) can be used to retrieve the error code.

See also: `FpSetRLimit` (230)

1.4.35 FpGetsid

Synopsis: Get current session ID.

Declaration: `function FpGetsid(pid: TPid) : TPid`

Visibility: default

Description: `FpGetsid` returns the session ID of the process `pid`. The return value is the session ID of the process. (it equals the PID of the session leader). The process `pid` must be in the same session as the current process.

Errors: On error, -1 is returned, and extended error information can be obtained with `fpGetErrno`.

See also: `FpGetpgid` (208), `FpGetpid` (209), `FpGetPpid` (209)

1.4.36 FpGetuid

Synopsis: Return current user ID.

Declaration: `function FpGetuid : TUid`

Visibility: default

Description: `FpGetuid` returns the real user ID of the currently running process.

Errors: None.

See also: `FpGetGid` (208), `FpGetEuid` (207), `FpGetEGid` (206), `FpGetPid` (209), `FpGetPPid` (209), `fpSetUID` (231)

Listing: `./examples/bunixex/ex17.pp`

Program Example17;

{ Program to demonstrate the GetUid and GetEuid functions. }

Uses BaseUnix;

begin

writeln ('User Id = ',fpgetuid,' Effective user Id = ',fpgeteuid);

end.

1.4.37 FpIOctl

Synopsis: General kernel IOCTL call.

Declaration: `function FpIOctl(Handle: cint; Ndx: TIOctlRequest; Data: Pointer) : cint`

Visibility: default

Description: This is a general interface to the Unix/ Linux `ioctl` call. It performs various operations on the file descriptor `Handle`. `Ndx` describes the operation to perform. `Data` points to data needed for the `Ndx` function. The structure of this data is function-dependent, so we don't elaborate on this here. For more information on this, see various manual pages under Linux.

Errors: Extended error information can be retrieved using `fpGetErrno` (207).

Listing: `./examples/bunixex/ex54.pp`

Program Example54;

uses BaseUnix,Termio;

{ Program to demonstrate the IOctl function. }

var

 tios : Termios;

begin

{ \$ifdef FreeBSD }

 fpIOctl(1,TIOCGETA,@tios); *// these constants are very OS dependant.*

// see the tcgetattr example for a better way

{ \$endif }

WriteLn('Input Flags : \$',hexstr(tios.c_iflag,8));

```

WriteLn('Output Flags : $',hexstr(tios.c_oflag,8));
WriteLn('Line Flags : $',hexstr(tios.c_lflag,8));
WriteLn('Control Flags: $',hexstr(tios.c_cflag,8));
end.

```

1.4.38 FpKill

Synopsis: Send a signal to a process.

Declaration: `function FpKill(pid: TPid; sig: cint) : cint`

Visibility: default

Description: `fpKill` sends a signal `Sig` to a process or process group. If `Pid`>0 then the signal is sent to `Pid`, if it equals -1, then the signal is sent to all processes except process 1. If `Pid`<-1 then the signal is sent to process group -`Pid`.

The return value is zero, except in case three, where the return value is the number of processes to which the signal was sent.

Errors: Extended error information can be retrieved using `fpGetErrno` (207):

sys_einval An invalid signal is sent.

sys_esrch The `Pid` or process group don't exist.

sys_eperm The effective userid of the current process doesn't math the one of process `Pid`.

See also: `FpSigAction` (232), `FpSignal` (234)

1.4.39 FpLink

Synopsis: Create a hard link to a file.

Declaration: `function FpLink(existing: PChar; newone: PChar) : cint`
`function FpLink(const existing: RawByteString;`
`const newone: RawByteString) : cint`

Visibility: default

Description: `fpLink` makes `NewOne` point to the same file as `Existing`. The two files then have the same inode number. This is known as a 'hard' link. The function returns zero if the call was successful, and returns a non-zero value if the call failed.

Errors: The following error codes are returned:

sys_exdev `Existing` and `NewOne` are not on the same file system.

sys_eperm The file system containing `Existing` and `NewOne` doesn't support linking files.

sys_eaccess Write access for the directory containing `NewOne` is disallowed, or one of the directories in `Existing` or `NewOne` has no search (=execute) permission.

sys_enoent A directory entry in `Existing` or `NewOne` does not exist or is a symbolic link pointing to a non-existent directory.

sys_enotdir A directory entry in `Existing` or `NewOne` is nor a directory.

sys_enomem Insufficient kernel memory.

sys_erofs The files are on a read-only file system.

sys_eexist NewOne already exists.

sys_emlink Existing has reached maximal link count.

sys_eloop existing or NewOne has a reference to a circular symbolic link, i.e. a symbolic link, whose expansion points to itself.

sys_enospc The device containing NewOne has no room for another entry.

sys_eperm Existing points to . or .. of a directory.

See also: [fpSymLink \(238\)](#), [fpUnLink \(243\)](#)

Listing: ./examples/bunixex/ex21.pp

Program Example21;

{ Program to demonstrate the Link and UnLink functions. }

Uses BaseUnix;

Var F : Text;

 S : String;

begin

 Assign (F,'test.txt');

Rewrite (F);

Writeln (F,'This is written to test.txt');

 Close(f);

{ new.txt and test.txt are now the same file }

if fpLink ('test.txt','new.txt')<>0 **then**

writeln ('Error when linking !');

{ Removing test.txt still leaves new.txt }

If fpUnlink ('test.txt')<>0 **then**

Writeln ('Error when unlinking !');

 Assign (f,'new.txt');

Reset (F);

While not EOF(f) **do**

begin

Readln(F,S);

Writeln ('> ',s);

end;

 Close (f);

{ Remove new.txt also }

If not FPUnlink ('new.txt')<>0 **then**

Writeln ('Error when unlinking !');

end.

1.4.40 FpLseek

Synopsis: Set file pointer position.

Declaration: function FpLseek (fd: cint; offset: TOff; whence: cint) : TOff

Visibility: default

Description: FpLseek sets the current file position of file fd to Offset, starting from Whence, which can be one of the following:

Seek_Set Offset is the absolute position in the file.

Seek_Cur Offset is relative to the current position.

Seek_end Offset is relative to the end of the file.

The function returns the new file position, or -1 of an error occurred.

For an example, see FpOpen (220).

Errors: Extended error information can be retrieved using fpGetErrno (207).

See also: FpOpen (220), FpWrite (245), FpClose (197), FpRead (224), FpFTruncate (205)

1.4.41 fpLstat

Synopsis: Return information about the symbolic link itself (doesn't follow the link).

Declaration: function fpLstat(path: PChar; Info: PStat) : cint
 function fpLstat(const path: RawByteString; Info: PStat) : cint
 function fpLstat(path: PChar; var Info: Stat) : cint
 function fpLstat(const Filename: RawByteString; var Info: Stat) : cint

Visibility: default

Description: FpLstat gets information about the link specified in Path (or FileName), and stores it in Info, which points to a record of type TStat. Contrary to FpFstat (204), it stores information about the link, not about the file the link points to. The function returns zero if the call was successful, a nonzero return value indicates failure.

Errors: Extended error information is returned by the FpGetErrno (207) function.

sys_enoent Path does not exist.

See also: FpFStat (204), #rtl.unixtype.TStatFS (2243)

Listing: ./examples/unixex/ex29.pp

```

program example29;

  { Program to demonstrate the LStat function. }

  uses BaseUnix, Unix;

  var f : text;
      i : byte;
      info : stat;

  begin
    { Make a file }
    assign (f, 'test.fil');
    rewrite (f);
    for i:=1 to 10 do writeln (f, 'Testline # ', i);
    close (f);
    { Do the call on made file. }
    if fpstat ('test.fil', info) <> 0 then
      begin
        writeln ('Fstat failed. Errno : ', fpgeterrno);
        halt (1);
      end;
    writeln;
  
```

```

writeln ('Result of stat on file "test.fil".');
writeln ('Inode : ',info.st_ino);
writeln ('Mode : ',info.st_mode);
writeln ('nlink : ',info.st_nlink);
writeln ('uid : ',info.st_uid);
writeln ('gid : ',info.st_gid);
writeln ('rdev : ',info.st_rdev);
writeln ('Size : ',info.st_size);
writeln ('Blksize : ',info.st_blksize);
writeln ('Blocks : ',info.st_blocks);
writeln ('atime : ',info.st_atime);
writeln ('mtime : ',info.st_mtime);
writeln ('ctime : ',info.st_ctime);

if fpSymLink ('test.fil','test.lnk')<>0 then
    writeln ('Link failed ! Errno : ',fpgeterrno);

if fplstat ('test.lnk',@info)<>0 then
    begin
        writeln('LStat failed. Errno : ',fpgeterrno);
        halt (1);
    end;
writeln;
writeln ('Result of fstat on file "test.lnk".');
writeln ('Inode : ',info.st_ino);
writeln ('Mode : ',info.st_mode);
writeln ('nlink : ',info.st_nlink);
writeln ('uid : ',info.st_uid);
writeln ('gid : ',info.st_gid);
writeln ('rdev : ',info.st_rdev);
writeln ('Size : ',info.st_size);
writeln ('Blksize : ',info.st_blksize);
writeln ('Blocks : ',info.st_blocks);
writeln ('atime : ',info.st_atime);
writeln ('mtime : ',info.st_mtime);
writeln ('ctime : ',info.st_ctime);
    { Remove file and link }
    erase (f);
    fpunlink ('test.lnk');
end.

```

1.4.42 FpMkdir

Synopsis: Create a new directory.

Declaration: `function FpMkdir(path: PChar; Mode: TMode) : cint`
`function FpMkdir(const path: RawByteString; Mode: TMode) : cint`

Visibility: default

Description: FpMkDir creates a new directory Path, and sets the new directory's mode to Mode. Path can be an absolute path or a relative path. Note that only the last element of the directory will be created, higher level directories must already exist, and must be writeable by the current user.

On success, 0 is returned. if the function fails, -1 is returned.

Note: There exist a portable alternative to fpMkDir: system.mkdir. Please use fpMkDir only if you are writing Unix specific code. System.mkdir will work on all operating systems.

Errors: Extended error information can be retrieved using `fpGetErrno` (207).

See also: `fpGetCWD` (206), `fpChDir` (195)

1.4.43 FpMkfifo

Synopsis: Create FIFO (named pipe) in file system.

Declaration: `function FpMkfifo(path: PChar; Mode: TMode) : cint`
`function FpMkfifo(const path: RawByteString; Mode: TMode) : cint`

Visibility: default

Description: `fpMkFifo` creates named a named pipe in the file system, with name `Path` and mode `Mode`.

The function returns zero if the command was successful, and nonzero if it failed.

Errors: The error codes include:

sys_enfile Too many file descriptors for this process.

sys_enfile The system file table is full.

1.4.44 Fpmmmap

Synopsis: Create memory map of a file.

Declaration: `function Fpmmmap(start: pointer; len: size_t; prot: cint; flags: cint; fd: cint; offst: off_t) : pointer`

Visibility: default

Description: `FpMMap` maps or unmaps files or devices into memory. The different arguments determine what and how the file is mapped:

adr Address where to `mmap` the device. This address is a hint, and may not be followed.

len Size (in bytes) of area to be mapped.

prot Protection of mapped memory. This is a OR-ed combination of the following constants:

PROT_EXEC The memory can be executed.

PROT_READ The memory can be read.

PROT_WRITE The memory can be written.

PROT_NONE The memory can not be accessed.

flags Contains some options for the `mmap` call. It is an OR-ed combination of the following constants:

MAP_FIXED Do not map at another address than the given address. If the address cannot be used, `MMap` will fail.

MAP_SHARED Share this map with other processes that map this object.

MAP_PRIVATE Create a private map with copy-on-write semantics.

MAP_ANONYMOUS `fd` does not have to be a file descriptor.

One of the options `MAP_SHARED` and `MAP_PRIVATE` must be present, but not both at the same time.

fd File descriptor from which to map.

off Offset to be used in file descriptor `fd`.

The function returns a pointer to the mapped memory, or a -1 in case of an error.

Errors: On error, -1 is returned and extended error information is returned by the FpGetErrno (207) function.

Sys_EBADF fd is not a valid file descriptor and MAP_ANONYMOUS was not specified.

Sys_EACCES MAP_PRIVATE was specified, but fd is not open for reading. Or MAP_SHARED was asked and PROT_WRITE is set, fd is not open for writing

Sys_EINVAL One of the record fields Start, length or offset is invalid.

Sys_ETXTBUSY MAP_DENYWRITE was set but the object specified by fd is open for writing.

Sys_EAGAIN fd is locked, or too much memory is locked.

Sys_ENOMEM Not enough memory for this operation.

See also: FpMUnMap (218)

Listing: ./examples/unixex/ex66.pp

Program Example66;

{ Program to demonstrate the MMap function. }

Uses BaseUnix, Unix;

Var S : String;

fd : cint;

Len : longint;

// args : tmmmapargs;

P : PChar;

begin

s:='This is the string';

Len:=Length(S);

fd:=fpOpen('testfile.txt',O_wrOnly or o_creat);

If fd=-1 **then**

Halt(1);

If fpWrite(fd,S[1],Len)=-1 **then**

Halt(2);

fpClose(fd);

fd:=fpOpen('testfile.txt',O_rdOnly);

if fd=-1 **then**

Halt(3);

P:=Pchar(fpmmmap(nil,len+1 ,PROT_READ or PROT_WRITE,MAP_PRIVATE,fd,0));

If longint(P)=-1 **then**

Halt(4);

Writeln('Read in memory :',P);

fpclose(fd);

if fpMUnMap(P,Len)<>0 **Then**

Halt(fpgeterrno);

end.

1.4.45 Fpmprotect

Synopsis: Set memory protection.

Declaration: function Fpmprotect(start: pointer; len: size_t; prot: cint) : cint

Visibility: default

Description: `fpmprotect` sets memory protection on a certain block of memory. The memory block to protect starts at `adr` and has size `len`. The `prot` argument specifies the kind of protection to apply, and is a bit-wise OR-ed combination of the following:

PROT_NONE Memory cannot be accessed

PROT_READ Memory can be read

PROT_WRITE Memory can be written

PROT_EXEC Memory can be executed

PROT_SEM Memory can be used for atomic operations

PROT_SAO Memory must have strong access order (powerPC only).

1.4.46 Fpmunmap

Synopsis: Unmap previously mapped memory block.

Declaration: `function Fpmunmap(start: pointer; len: size_t) : cint`

Visibility: default

Description: `FpMUnMap` unmaps the memory block of size `Len`, pointed to by `Adr`, which was previously allocated with `FpMMap` (216).

The function returns `True` if successful, `False` otherwise.

For an example, see `FpMMap` (216).

Errors: In case of error the function returns a nonzero value, extended error information is returned by the `FpGetErrno` (207) function. See `FpMMap` (216) for possible error values.

See also: `FpMMap` (216)

1.4.47 FpNanoSleep

Synopsis: Suspend process for a short time.

Declaration: `function FpNanoSleep(req: ptimespec; rem: ptimespec) : cint`

Visibility: default

Description: `FpNanoSleep` suspends the process till a time period as specified in `req` has passed. Then the function returns. If the call was interrupted (e.g. by some signal) then the function may return earlier, and `rem` will contain the remaining time till the end of the intended period. In this case the return value will be -1, and `ErrNo` will be set to `EINTR`.

If the function returns without error, the return value is zero.

Errors: If an error occurred or the call was interrupted, -1 is returned. Extended error information can be retrieved using `fpGetErrno` (207).

See also: `FpPause` (222), `FpAlarm` (194)

Listing: `./examples/bunixex/ex72.pp`

```

program example72;

{ Program to demonstrate the NanoSleep function. }

uses BaseUnix;

Var
  Req,Rem : TimeSpec;
  Res : Longint;

begin
  With Req do
    begin
      tv_sec:=10;
      tv_nsec:=100;
    end;
  Write('NanoSleep returned : ');
  Flush(Output);
  Res:=(fpNanoSleep(@Req,@rem));
  WriteLn(res);
  If (res<>0) then
    With rem do
      begin
        WriteLn('Remaining seconds : ',tv_sec);
        WriteLn('Remaining nanoseconds : ',tv_nsec);
      end;
end.

```

1.4.48 fpNice

Synopsis: Set process priority.

Declaration: `function fpNice(N: cint) : cint`

Visibility: default

Description: `Nice` adds `-N` to the priority of the running process. The lower the priority numerically, the less the process is favored. Only the superuser can specify a negative `N`, i.e. increase the rate at which the process is run.

If the function is successful, zero is returned. On error, a nonzero value is returned.

Errors: Extended error information is returned by the `FpGetErrno` (207) function.

sys_eperm A non-superuser tried to specify a negative `N`, i.e. do a priority increase.

See also: `FpGetPriority` (210), `FpSetPriority` (230)

Listing: `./examples/unixex/ex15.pp`

```

Program Example15;

{ Program to demonstrate the Nice and Get/SetPriority functions. }

Uses BaseUnix,Unix;

begin

```

```

writeln ('Setting priority to 5');
fpsetpriority (prio_process,fpgetpid,5);
writeln ('New priority = ',fpgetpriority (prio_process,fpgetpid));
writeln ('Doing nice 10');
fpnice (10);
writeln ('New Priority = ',fpgetpriority (prio_process,fpgetpid));
end.

```

1.4.49 FpOpen

Synopsis: Open file and return file descriptor.

Declaration:

```

function FpOpen(path: PChar; flags: cint; Mode: TMode) : cint
function FpOpen(path: PChar; flags: cint) : cint
function FpOpen(const path: RawByteString; flags: cint) : cint
function FpOpen(const path: RawByteString; flags: cint; Mode: TMode)
    : cint
function FpOpen(path: ShortString; flags: cint) : cint
function FpOpen(path: ShortString; flags: cint; Mode: TMode) : cint

```

Visibility: default

Description: FpOpen opens a file in Path with flags flags and mode Mode One of the following:

O_RdOnly File is opened Read-only

O_WrOnly File is opened Write-only

O_RdWr File is opened Read-Write

The flags may beOR-ed with one of the following constants:

O_Creat File is created if it doesn't exist.

O_Excl If the file is opened with O_Creat and it already exists, the call will fail.

O_NoCtty If the file is a terminal device, it will NOT become the process' controlling terminal.

O_Trunc If the file exists, it will be truncated.

O_Append the file is opened in append mode. *Before each write*, the file pointer is positioned at the end of the file.

O_NonBlock The file is opened in non-blocking mode. No operation on the file descriptor will cause the calling process to wait till.

O_NDelay Idem as O_NonBlock

O_Sync The file is opened for synchronous IO. Any write operation on the file will not return until the data is physically written to disk.

O_NoFollow if the file is a symbolic link, the open fails. (Linux 2.1.126 and higher only)

O_Directory if the file is not a directory, the open fails. (Linux 2.1.126 and higher only)

Path can be of type PChar or String. The optional mode argument specifies the permissions to set when opening the file. This is modified by the umask setting. The real permissions are Mode and not umask. The return value of the function is the file descriptor, or a negative value if there was an error.

Errors: Extended error information can be retrieved using fpGetErrno ([207](#)).

See also: FpClose ([197](#)), FpRead ([224](#)), FpWrite ([245](#)), FpFTruncate ([205](#)), FpLSeek ([213](#))

Listing: ./examples/bunixex/ex19.pp

Program Example19;

{ Program to demonstrate the fpOpen, fpwrite and fpClose functions. }

Uses BaseUnix;

Const Line : **String**[80] = 'This is easy writing !';

Var FD : **Cint**;

begin

FD:=fpOpen ('Test.dat',O_WrOnly or O_Creat);

if FD>0 **then**

begin

if length(Line)<>fpwrite (FD,Line[1],Length(Line)) **then**

Writeln ('Error when writing to file !');

fpClose(FD);

end;

end.

1.4.50 FpOpendir

Synopsis: Open a directory for reading.

Declaration: function FpOpendir(dirname: PChar) : pDir
 function FpOpendir(const dirname: RawByteString) : pDir
 function FpOpendir(dirname: ShortString) : pDir

Visibility: default

Description: FpOpenDir opens the directory DirName, and returns a pdir pointer to a Dir (247) record, which can be used to read the directory structure. If the directory cannot be opened, nil is returned.

Errors: Extended error information can be retrieved using fpGetErrno (207).

See also: FpCloseDir (198), FpReadDir (225)

Listing: ./examples/bunixex/ex35.pp

Program Example35;

*{ Program to demonstrate the
 OpenDir, ReadDir, SeekDir and TellDir functions. }*

Uses BaseUnix;

Var TheDir : PDir;

ADirent : PDirent;

Entry : Longint;

begin

TheDir:=fpOpenDir('./.');

Repeat

// Entry:=fpTellDir(TheDir);

ADirent:=fpReadDir (TheDir^);

If ADirent<>Nil **then**

```

    With ADirent^ do
    begin
        Writeln ('Entry No : ',Entry);
        Writeln ('Inode : ',d_fileno);
        // Writeln ('Offset : ',d_off);
        Writeln ('Reclen : ',d_reclen);
        Writeln ('Name : ',pchar(@d_name[0]));
    end;
Until ADirent=Nil;
Repeat
    Write ('Entry No. you would like to see again (-1 to stop): ');
    ReadLn (Entry);
    If Entry<>-1 then
    begin
        // fpSeekDir (TheDir,Entry); // not implemented for various platforms
        ADirent:=fpReadDir (TheDir^);
        If ADirent<>Nil then
        With ADirent^ do
        begin
            Writeln ('Entry No : ',Entry);
            Writeln ('Inode : ',d_fileno);
            // Writeln ('Offset : ',off);
            Writeln ('Reclen : ',d_reclen);
            Writeln ('Name : ',pchar(@d_name[0]));
        end;
    end;
Until Entry=-1;
fpCloseDir (TheDir^);
end.

```

1.4.51 FpPause

Synopsis: Wait for a signal to arrive.

Declaration: `function FpPause : cint`

Visibility: default

Description: `FpPause` puts the process to sleep and waits until the application receives a signal. If a signal handler is installed for the received signal, the handler will be called and after that pause will return control to the process.

For an example, see `fpAlarm` ([194](#)).

1.4.52 FpPipe

Synopsis: Create a set of pipe file handlers.

Declaration: `function FpPipe(var fildes: TFilDes) : cint`

Visibility: default

Description: `FpPipe` creates a pipe, i.e. two file objects, one for input, one for output. The file handles are returned in the array `fildes`. The input handle is in the 0-th element of the array, the output handle is in the 1-st element.

The function returns zero if everything went successfully, a nonzero return value indicates an error.

Errors: In case the function fails, the following return values are possible:

sys_enfile Too many file descriptors for this process.

sys_enfile The system file table is full.

See also: `#rtl.unix.POpen` (2219), `fpMkFifo` (216)

Listing: `./examples/bunixex/ex36.pp`

Program Example36;

{ Program to demonstrate the AssignPipe function. }

Uses BaseUnix, Unix;

Var pipi, pipo : Text;
s : String;

```
begin
  Writeln ('Assigning Pipes. ');
  If assignpipe(pipi, pipo) <> 0 then
    Writeln ('Error assigning pipes !', fpgeterrno);
  Writeln ('Writing to pipe, and flushing. ');
  Writeln (pipo, 'This is a textstring'); close(pipo);
  Writeln ('Reading from pipe. ');
  While not eof(pipi) do
    begin
      Readln (pipi, s);
      Writeln ('Read from pipe : ', s);
    end;
  close (pipi);
  writeln ('Closed pipes. ');
  writeln
end.
```

1.4.53 FpPoll

Synopsis: Poll a file descriptor for events.

Declaration: `function FpPoll(fds: ppollfd; nfds: cuint; timeout: clong) : cint`

Visibility: default

Description: `fpPoll` waits for events on file descriptors. `fds` points to an array of `tpollfd` records, each of these records describes a file descriptor on which to wait for events. The number of file descriptors is given by `nfds`. `>timeout` specifies the maximum time (in milliseconds) to wait for events.

On timeout, the result value is 0. If an event occurred on some descriptors, then the return value is the number of descriptors on which an event (or error) occurred. The `revents` field of the `tpollfd` records will contain the events for the file descriptor it described.

See also: `tpollfd` (191)

1.4.54 FppRead

Synopsis: Positional read: read from file descriptor at a certain position.

Declaration: `function FpPRead(fd: cint; buf: PChar; nbytes: TSize; offset: TOff) : TsSize`
`function FppRead(fd: cint; var buf; nbytes: TSize; offset: TOff) : TsSize`

Visibility: default

Description: `FpPRead` reads `nbytes` bytes from file descriptor `fd` into buffer `buf` starting at offset `offset`. Offset is measured from the start of the file. This function can only be used on files, not on pipes or sockets (i.e. any seekable file descriptor).

The function returns the number of bytes actually read, or -1 on error.

Errors: On error, -1 is returned.

See also: `FpReadV` (227), `FpPWrite` (224)

1.4.55 FppWrite

Synopsis: Positional write: write to file descriptor at a certain position.

Declaration: `function FpPWrite(fd: cint; buf: PChar; nbytes: TSize; offset: TOff) : TsSize`
`function FppWrite(fd: cint; const buf; nbytes: TSize; offset: TOff) : TsSize`

Visibility: default

Description: `FpPWrite` writes `nbytes` bytes from buffer `buf` into file descriptor `fd` starting at offset `offset`. Offset is measured from the start of the file. This function can only be used on files, not on pipes or sockets (i.e. any seekable file descriptor).

The function returns the number of bytes actually written, or -1 on error.

Errors: On error, -1 is returned.

See also: `FpPRead` (223), `FpWriteV` (246)

1.4.56 FpRead

Synopsis: Read data from file descriptor.

Declaration: `function FpRead(fd: cint; buf: PChar; nbytes: TSize) : TsSize`
`function FpdRead(fd: cint; var buf; nbytes: TSize) : TsSize`

Visibility: default

Description: `FpdRead` reads at most `nbytes` bytes from the file descriptor `fd`, and stores them in `buf`.

The function returns the number of bytes actually read, or -1 if an error occurred. No checking on the length of `buf` is done.

Errors: Extended error information can be retrieved using `fpGetErrno` (207).

See also: `FpOpen` (220), `FpClose` (197), `FpWrite` (245), `FpFTruncate` (205), `FpLSeek` (213)

Listing: `./examples/bunixex/ex20.pp`

Program Example20;*{ Program to demonstrate the fdRead and fdTruncate functions. }***Uses** BaseUnix;**Const** Data : **string**[10] = '1234567890';**Var** FD : cint;
 I : longint;**begin**

FD:=fpOpen('test.dat',o_wronly or o_creat,&666);

if fd>0 **then** **begin** *{ Fill file with data }* **for** I:=1 **to** 10 **do** **if** fpWrite (FD,Data[I],10)<>10 **then** **begin** **writeln** ('Error when writing !'); **halt**(1); **end**;

fpClose(FD);

FD:=fpOpen('test.dat',o_rdonly);

{ Read data again } **if** FD>0 **then** **begin** **For** I:=1 **to** 5 **do** **if** fpRead (FD,Data[I],10)<>10 **then** **begin** **Writeln** ('Error when Reading !'); **Halt**(2); **end**;

fpClose(FD);

{ Truncating file at 60 bytes } *{ For truncating, file must be open or write }*

FD:=fpOpen('test.dat',o_wronly,&666);

if FD>0 **then** **begin** **if** fpfTruncate(FD,60)<>0 **then** **Writeln**('Error when truncating !');

fpClose (FD);

end; **end**; **end**;**end.**

1.4.57 FpReaddir

Synopsis: Read entry from directory.

Declaration: **function** FpReaddir(**var** dirp: Dir) : pDirent

Visibility: default

Description: FpReadDir reads the next entry in the directory pointed to by dirp. It returns a pdirent pointer to a dirent (248) record describing the entry. If the next entry can't be read, Nil is returned.

For an example, see `FpOpenDir` (221).

Errors: Extended error information can be retrieved using `fpGetErrno` (207).

See also: `FpCloseDir` (198), `FpOpenDir` (221)

1.4.58 `fpReadLink`

Synopsis: Read destination of symbolic link.

Declaration: `function fpReadLink(name: PChar; linkname: PChar; maxlen: size_t) : cint`
`function fpReadLink(const Name: RawByteString) : RawByteString`

Visibility: default

Description: `fpReadLink` returns the file the symbolic link `name` is pointing to. The first form of this function accepts a buffer `linkname` of length `maxlen` where the filename will be stored. It returns the actual number of characters stored in the buffer.

The second form of the function returns simply the name of the file.

Errors: On error, the first form of the function returns -1; the second one returns an empty string. Extended error information is returned by the `FpGetErrno` (207) function.

SYS_ENOTDIR A part of the path in `Name` is not a directory.

SYS_EINVAL `maxlen` is not positive, or the file is not a symbolic link.

SYS_ENAMETOOLONG A pathname, or a component of a pathname, was too long.

SYS_ENOENT the link `name` does not exist.

SYS_EACCES No permission to search a directory in the path

SYS_ELOOP Too many symbolic links were encountered in translating the pathname.

SYS_EIO An I/O error occurred while reading from the file system.

SYS_EFAULT The buffer is not part of the process's memory space.

SYS_ENOMEM Not enough kernel memory was available.

See also: `FpSymLink` (238)

Listing: `./examples/unixex/ex62.pp`

Program Example62;

{ Program to demonstrate the ReadLink function. }

Uses BaseUnix, Unix;

Var F : Text;
 S : String;

begin
 Assign (F, 'test.txt');
 Rewrite (F);
 Writeln (F, 'This is written to test.txt');
 Close(f);
 { new.txt and test.txt are now the same file }
 if fpSymLink ('test.txt', 'new.txt') <> 0 **then**
 writeln ('Error when symlinking !');
 S := fpReadLink('new.txt');

```

If S=" then
  Writeln ('Error reading link !')
Else
  Writeln ('Link points to : ',S);
  { Now remove links }
If fpUnlink ('new.txt')<>0 then
  Writeln ('Error when unlinking !');
If fpUnlink ('test.txt')<>0 then
  Writeln ('Error when unlinking !');
end.

```

1.4.59 FpReadV

Synopsis: Vector read: Read into multiple buffers.

Declaration: `function FpReadV(fd: cint; const iov: piovec; iovcnt: cint) : TsSize`

Visibility: default

Description: `FpReadV` reads data from file descriptor `fd` and writes it into `iovcnt` buffers described by the `iovec` (191) buffers pointed to by `iov`. It works like `fpRead` (224) only on multiple buffers.

Errors: On error, -1 is returned.

See also: `FpWriteV` (246), `FpPWrite` (224), `FpPRead` (223)

1.4.60 FpRename

Synopsis: Rename file.

Declaration: `function FpRename(old: PChar; newpath: PChar) : cint`
`function FpRename(const old: RawByteString;`
`const newpath: RawByteString) : cint`

Visibility: default

Description: `FpRename` renames the file `Old` to `NewPath`. `NewPath` can be in a different directory than `Old`, but it cannot be on another partition (device). Any existing file on the new location will be replaced.

If the operation fails, then the `Old` file will be preserved.

The function returns zero on success, a nonzero value indicates failure.

Note: There exist a portable alternative to `fpRename`: `system.rename`. Please use `fpRename` only if you are writing Unix specific code. `System.rename` will work on all operating systems.

Errors: Extended error information can be retrieved using `fpGetErrno` (207).

sys_eisdir `NewPath` exists and is a directory, but `Old` is not a directory.

sys_exdev `NewPath` and `Old` are on different devices.

sys_enotempty or sys_eexist `NewPath` is an existing, non-empty directory.

sys_ebusy `Old` or `NewPath` is a directory and is in use by another process.

sys_einval `NewPath` is part of `Old`.

sys_emlink `OldPath` or `NewPath` already have the maximum amount of links pointing to them.

sys_enotdir part of `Old` or `NewPath` is not directory.

sys_efault For the `pchar` case: One of the pointers points to an invalid address.

sys_eaccess access is denied when attempting to move the file.

sys_enametoolong Either `Old` or `NewPath` is too long.

sys_enoent a directory component in `Old` or `NewPath` didn't exist.

sys_enomem not enough kernel memory.

sys_erofs `NewPath` or `Old` is on a read-only file system.

sys_eloop too many symbolic links were encountered trying to expand `Old` or `NewPath`

sys_enospc the file system has no room for the new directory entry.

See also: `FpUnLink` (243)

1.4.61 FpRmdir

Synopsis: Remove a directory.

Declaration: `function FpRmdir(path: PChar) : cint`
`function FpRmdir(const path: RawByteString) : cint`

Visibility: default

Description: `FpRmdir` removes the directory `Path` from the system. The directory must be empty for this call to succeed, and the user must have the necessary permissions in the parent directory. Only the last component of the directory is removed, i.e. higher-lying directories are not removed.

On success, zero is returned. A nonzero return value indicates failure.

Note: There exist a portable alternative to `fpRmdir`: `system.rmdir`. Please use `fpRmdir` only if you are writing Unix specific code. `System.rmdir` will work on all operating systems.

Errors: Extended error information can be retrieved using `fpGetErrno` (207).

1.4.62 fpSelect

Synopsis: Wait for events on file descriptors.

Declaration: `function FPSelect(N: cint; readfds: pFDSet; writefds: pFDSet;`
`exceptfds: pFDSet; Timeout: ptimeval) : cint`
`function fpSelect(N: cint; readfds: pFDSet; writefds: pFDSet;`
`exceptfds: pFDSet; Timeout: cint) : cint`
`function fpSelect(var T: Text; Timeout: ptimeval) : cint`
`function fpSelect(var T: Text; Timeout: time_t) : cint`

Visibility: default

Description: `FpSelect` checks one of the file descriptors in the `FDSet`s to see if the following I/O operation on the file descriptors will block.

`readfds`, `writefds` and `exceptfds` are pointers to arrays of 256 bits. If you want a file descriptor to be checked, you set the corresponding element in the array to 1. The other elements in the array must be set to zero. Three arrays are passed : The entries in `readfds` are checked to see if the following read operation will block. The entries in `writefds` are checked to see if the following write operation will block, while entries in `exceptfds` are checked to see if an exception occurred on them.

You can use the functions `fpFD_ZERO` (204), `fpFD_Clr` (203), `fpFD_Set` (203) or `fpFD_IsSet` (203) to manipulate the individual elements of a set.

The pointers can be `Nil`.

N is the value of the largest file descriptor in one of the sets, + 1. In other words, it is the position of the last bit which is set in the array of bits.

TimeOut can be used to set a time limit. If TimeOut can be two types :

1. TimeOut is of type `ptimeval` and contains a zero time, the call returns immediately. If TimeOut is `Nil`, the kernel will wait forever, or until a status changed.
2. TimeOut is of type `cint`. If it is -1, this has the same effect as a Timeout of type `PTime` which is `Nil`. Otherwise, TimeOut contains a time in milliseconds.

When the TimeOut is reached, or one of the file descriptors has changed, the `Select` call returns. On return, it will have modified the entries in the array which have actually changed, and it returns the number of entries that have been changed. If the timeout was reached, and no descriptor changed, zero is returned; The arrays of indexes are undefined after that. On error, -1 is returned.

The variant with the text file will execute the `FpSelect` call on the file descriptor associated with the text file `T`

Errors: On error, the function returns -1. Extended error information can be retrieved using `fpGetErrno` (207).

SYS_EBADF An invalid descriptor was specified in one of the sets.

SYS_EINTR A non blocked signal was caught.

SYS_EINVAL N is negative or too big.

SYS_ENOMEM `Select` was unable to allocate memory for its internal tables.

See also: `fpFD_ZERO` (204), `fpFD_Clr` (203), `fpFD_Set` (203), `fpFD_IsSet` (203)

Listing: `./examples/bunixex/ex33.pp`

Program Example33;

{ Program to demonstrate the Select function. }

Uses BaseUnix;

Var FDS : `Tfdset`;

begin

```

    fpfd_zero(FDS);
    fpfd_set(0,FDS);
    Writeln ('Press the <ENTER> to continue the program. ');
    { Wait until File descriptor 0 (=Input) changes }
    fpSelect (1,@FDS,nil,nil,nil);
    { Get rid of <ENTER> in buffer }
    readln;
    Writeln ('Press <ENTER> key in less than 2 seconds... ');
    Fpfd_zero(FDS);
    FpFd_set (0,FDS);
    if fpSelect (1,@FDS,nil,nil,2000)>0 then
        Writeln ('Thank you !')
        { FD_ISSET(0,FDS) would be true here. }

```

```

    else
        Writeln ('Too late !');
    end.

```

1.4.63 fpseterrno

Synopsis: Set extended error information.

Declaration: `procedure fpseterrno(err: LongInt)`

Visibility: default

Description: `fpseterrno` sets the extended information on the latest error. It is called by all functions that communicate with the kernel or C library.

Unless a direct kernel call is performed, there should never be any need to call this function.

See also: `fpgeterrno` (207)

1.4.64 FpSetgid

Synopsis: Set the current group ID.

Declaration: `function FpSetgid(gid: TGid) : cint`

Visibility: default

Description: `fpSetUID` sets the group ID of the current process. This call will only work if it is executed as root, or the program is setgid root.

On success, zero is returned, on error -1 is returned.

Errors: Extended error information can be retrieved with `fpGetErrNo` (207).

See also: `FpSetUid` (231), `FpGetGid` (208), `FpGetUid` (211), `FpGetEUid` (207), `FpGetEGid` (206), `FpGetPid` (209), `FpGetPPid` (209)

1.4.65 fpSetPriority

Synopsis: Set process priority.

Declaration: `function fpSetPriority(Which: cint; Who: cint; What: cint) : cint`

Visibility: default

Description: `fpSetPriority` sets the priority with which a process is running. Which process(es) is determined by the `Which` and `Who` variables. Which can be one of the predefined constants:

Prio_Process `Who` is interpreted as process ID

Prio_PGrp `Who` is interpreted as process group ID

Prio_User `Who` is interpreted as user ID

`Prio` is a value in the range -20 to 20.

For an example, see `FpNice` (219).

The function returns zero on success, -1 on failure

Errors: Extended error information is returned by the `FpGetErrno` (207) function.

sys_esrch No process found using `which` and `who`.

sys_einval `Which` was not one of `Prio_Process`, `Prio_Grp` or `Prio_User`.

sys_eperm A process was found, but neither its effective or real user ID match the effective user ID of the caller.

sys_eacces A non-superuser tried to a priority increase.

See also: `FpGetPriority` (210), `FpNice` (219)

1.4.66 FpSetRLimit

Synopsis: Set process resource limits.

Declaration: `function FpSetRLimit(Resource: cint; rlim: PRLimit) : cint`

Visibility: default

Description: `FpGetRLimit` sets the resource limits for the current process: `resource` determines the resource of which the kernel should set the limits (one of the many `RLIMIT_*` constants). `rlim` should point to a `TRLimit` (252) record which contains the new limits for the resource indicated in `resource`.

The function returns zero if the resource limits were successfully set.

Errors: On error, -1 is returned and `fpgeterrno` (207) can be used to retrieve the error code.

See also: `FpGetRLimit` (210)

1.4.67 FpSetsid

Synopsis: Create a new session.

Declaration: `function FpSetsid : TPid`

Visibility: default

Description: `FpSetsid` creates a new session (process group). It returns the new process group id (as returned by `FpGetgrp` (208)). This call will fail if the current process is already the process group leader.

Errors: On error, -1 is returned. Extended error information can be retrieved with `fpGetErrNo` (207)

1.4.68 fpsettimeofday

Synopsis: Set kernel time.

Declaration: `function fpsettimeofday(tp: ptimeval; tzp: ptimezone) : cint`

Visibility: default

Description: `FpSetTimeOfDay` sets the kernel time to the number of seconds since 00:00, January 1 1970, GMT specified in the `tp` record. This time NOT corrected any way, not taking into account time-zones, daylight savings time and so on.

It is simply a wrapper to the kernel system call.

See also: `#rtl.unix.FPGetTimeOfDay` (2214)

1.4.69 FpSetuid

Synopsis: Set the current user ID.

Declaration: `function FpSetuid(uid: TUid) : cint`

Visibility: default

Description: `fpSetUID` sets the user ID of the current process. This call will only work if it is executed as root, or the program is `setuid root`.

On success, zero is returned, on error -1 is returned.

Errors: Extended error information can be retrieved with `fpGetErrNo` (207).

See also: `FpGetGid` (208), `FpGetUid` (211), `FpGetEUid` (207), `FpGetEGid` (206), `FpGetPid` (209), `FpGetPPid` (209), `FpSetGid` (230)

1.4.70 FPSigaction

Synopsis: Install signal handler.

Declaration: `function FPSigaction(sig: cint; act: psigactionrec; oact: psigactionrec)
: cint`

Visibility: default

Description: `FPSigaction` changes the action to take upon receipt of a signal. `Act` and `Oact` are pointers to a `SigActionRec` (249) record. `Sig` specifies the signal, and can be any signal except **SIGKILL** or **SIGSTOP**.

If `Act` is non-nil, then the new action for signal `Sig` is taken from it. If `Oact` is non-nil, the old action is stored there. `Sa_Handler` may be `SIG_DFL` for the default action or `SIG_IGN` to ignore the signal. `Sa_Mask` Specifies which signals should be ignored during the execution of the signal handler. `Sa_Flags` Specifies a series of flags which modify the behaviour of the signal handler. You can 'or' none or more of the following :

SA_NOCLDSTOP If `sig` is **SIGCHLD** do not receive notification when child processes stop.

SA_ONESHOT or **SA_RESETHAND** Restore the signal action to the default state once the signal handler has been called.

SA_RESTART For compatibility with BSD signals.

SA_NOMASK or **SA_NODEFER** Do not prevent the signal from being received from within its own signal handler.

Errors: Extended error information can be retrieved using `fpGetErrno` (207).

sys_einval an invalid signal was specified, or it was **SIGKILL** or **SIGSTOP**.

sys_efault `Act`, `OldAct` point outside this process address space

sys_eintr System call was interrupted.

See also: `FpSigProcMask` (235), `FpSigPending` (235), `FpSigSuspend` (236), `FpKill` (212)

Listing: `./examples/bunixex/ex57.pp`

Program `example57;`

{ Program to demonstrate the SigAction function. }

```
{  
do a kill -USR1 pid from another terminal to see what happens.  
replace pid with the real pid of this program.  
You can get this pid by running 'ps'.  
}
```

uses `BaseUnix;`

Var

`oa,na : PSigActionRec;`

Procedure `DoSig(sig : cint);cdecl;`

begin

`writeln('Receiving signal: ',sig);`

end;

```

begin
  new(na);
  new(oa);
  na^.sa_Handler:=SigActionHandler(@DoSig);
  fillchar(na^.Sa_Mask,sizeof(na^.sa_mask),#0);
  na^.Sa_Flags:=0;
  {$ifdef Linux} // Linux specific
    na^.Sa_Restorer:=Nil;
  {$endif}
  if fpSigAction(SigUsr1,na,oa)<>0 then
    begin
      writeln('Error: ',fpgeterrno,');
      halt(1);
    end;
  Writeln ('Send USR1 signal or press <ENTER> to exit');
  readln;
end.

```

1.4.71 FpSigAddSet

Synopsis: Set a signal in a signal set.

Declaration: `function FpSigAddSet (var nset: tsigset; signo: cint) : cint`

Visibility: default

Description: `FpSigAddSet` adds signal `Signo` to the signal set `nset`. The function returns 0 on success.

Errors: If an invalid signal number is given, -1 is returned.

See also: `FpSigEmptySet` ([233](#)), `FpSigFillSet` ([234](#)), `FpSigDelSet` ([233](#)), `FpSigIsMember` ([234](#))

1.4.72 FpSigDelSet

Synopsis: Remove a signal from a signal set.

Declaration: `function FpSigDelSet (var nset: tsigset; signo: cint) : cint`

Visibility: default

Description: `FpSigDelSet` removes signal `Signo` to the signal set `nset`. The function returns 0 on success.

Errors: If an invalid signal number is given, -1 is returned.

See also: `FpSigEmptySet` ([233](#)), `FpSigFillSet` ([234](#)), `FpSigAddSet` ([233](#)), `FpSigIsMember` ([234](#))

1.4.73 FpsigEmptySet

Synopsis: Clear all signals from signal set.

Declaration: `function FpsigEmptySet (var nset: tsigset) : cint`

Visibility: default

Description: `FpSigEmptySet` clears all signals from the signal set `nset`.

Errors: None. This function always returns zero.

See also: `FpSigFillSet` ([234](#)), `FpSigAddSet` ([233](#)), `FpSigDelSet` ([233](#)), `FpSigIsMember` ([234](#))

1.4.74 FpSigFillSet

Synopsis: Set all signals in signal set.

Declaration: `function FpSigFillSet (var nset: tsigset) : cint`

Visibility: default

Description: `FpSigFillSet` sets all signals in the signal set `nset`.

Errors: None. This function always returns zero.

See also: `FpSigEmptySet` (233), `FpSigAddSet` (233), `FpSigDelSet` (233), `FpSigIsMember` (234)

1.4.75 FpSigIsMember

Synopsis: Check whether a signal appears in a signal set.

Declaration: `function FpSigIsMember (const nset: tsigset; signo: cint) : cint`

Visibility: default

Description: `FpSigIsMember` checks whether `SigNo` appears in the set `nset`. If it is a member, then 1 is returned. If not, zero is returned.

Errors: If an invalid signal number is given, -1 is returned.

See also: `FpSigEmptySet` (233), `FpSigFillSet` (234), `FpSigAddSet` (233), `FpSigDelSet` (233)

1.4.76 FpSignal

Synopsis: Install signal handler (deprecated).

Declaration: `function FpSignal (signum: LongInt; Handler: signalhandler)
: signalhandler`

Visibility: default

Description: `FpSignal` installs a new signal handler (specified by `Handler`) for signal `SigNum`.

This call has a subset of the functionality provided by the `FpSigAction` (232) call. The return value for `FpSignal` is the old signal handler, or nil on error.

Errors: Extended error information can be retrieved using `fpGetErrno` (207).

SIG_ERR An error occurred.

See also: `FpSigAction` (232), `FpKill` (212)

Listing: `./examples/bunixex/ex58.pp`

Program `example58;`

{ Program to demonstrate the Signal function. }

```
{
do a kill -USR1 pid from another terminal to see what happens.
replace pid with the real pid of this program.
You can get this pid by running 'ps'.
}
```

```

uses BaseUnix;

Procedure DoSig(sig : cint);cdecl;

begin
    writeln('Receiving signal: ',sig);
end;

begin
    if fpSignal(SigUsr1,SignalHandler(@DoSig))=signalhandler(SIG_ERR) then
        begin
            writeln('Error: ',fpGetErrno,'. ');
            halt(1);
        end;
    Writeln ('Send USR1 signal or press <ENTER> to exit');
    readln;
end.

```

1.4.77 FpSigPending

Synopsis: Return set of currently pending signals.

Declaration: `function FpSigPending(var nset: tsigset) : cint`

Visibility: default

Description: `fpSigpending` allows the examination of pending signals (which have been raised while blocked.)
The signal mask of pending signals is returned.

Errors: None

See also: `fpSigAction` (232), `fpSigProcMask` (235), `fpSigSuspend` (236), `fpSignal` (234), `fpKill` (212)

1.4.78 FpSigProcMask

Synopsis: Set list of blocked signals.

Declaration: `function FpSigProcMask(how: cint; nset: psigset; oset: psigset) : cint`
`function FpSigProcMask(how: cint; constref nset: tsigset;`
`var oset: tsigset) : cint`

Visibility: default

Description: Changes the list of currently blocked signals. The behaviour of the call depends on `How` :

SIG_BLOCK The set of blocked signals is the union of the current set and the `nset` argument.

SIG_UNBLOCK The signals in `nset` are removed from the set of currently blocked signals.

SIG_SETMASK The list of blocked signals is set so `nset`.

If `oset` is non-nil, then the old set is stored in it.

Errors: `Errno` is used to report errors.

sys_efault `oset` or `nset` point to an address outside the range of the process.

sys_eintr System call was interrupted.

See also: `fpSigAction` (232), `fpSigPending` (235), `fpSigSuspend` (236), `fpKill` (212)

1.4.79 FpSigSuspend

Synopsis: Set signal mask and suspend process till signal is received.

Declaration: `function FpSigSuspend(const sigmask: tsigset) : cint`

Visibility: default

Description: `FpSigSuspend` temporarily replaces the signal mask for the process with the one given in `SigMask`, and then suspends the process until a signal is received.

Errors: None

See also: `FpSigAction` (232), `FpSigProcMask` (235), `FpSigPending` (235), `FpSignal` (234), `FpKill` (212)

1.4.80 FpSigTimedWait

Synopsis: Wait for signal, with timeout.

Declaration: `function FpSigTimedWait(const sigset: tsigset; info: psiginfo; timeout: ptimespec) : cint`

Visibility: default

Description: `FpSigTimedWait` will suspend the current thread and wait for one of the signals in `sigset` to be delivered. information on the delivered signal is placed in the location provided by `info` (or in `info` itself, if the `Var` variant of the call is used). If the signal is not delivered within the time limit set in `timeout`, then the call will return -1, and `FpGetErrno` will return `EAGAIN`.

On success, the signal number is returned.

Errors: On error, -1 is returned, and extended error information can be obtained with `FpGetErrno`.

See also: `FpSigSuspend` (236)

1.4.81 FpSleep

Synopsis: Suspend process for several seconds.

Declaration: `function FpSleep(seconds: cuint) : cuint`

Visibility: default

Description: `FpSleep` suspends the process till a time period as specified in `seconds` has passed, then the function returns. If the call was interrupted (e.g. by some signal) then the function may return earlier, and the return value is the remaining time till the end of the intended period.

If the function returns without error, the return value is zero.

See also: `FpPause` (222), `FpAlarm` (194), `FpNanoSleep` (218)

Listing: `./examples/bunixex/ex73.pp`

program `example73;`

{ Program to demonstrate the FpSleep function. }

uses `BaseUnix;`

Var

```

Res : Longint;

begin
  Write('Sleep returned : ');
  Flush(Output);
  Res:=(fpSleep(10));
  Writeln(res);
  If (res<>0) then
    Writeln('Remaining seconds : ',res);
end.

```

1.4.82 FpStat

Synopsis: Retrieve file information about a file descriptor.

Declaration: `function FpStat (path: PChar; var buf: Stat) : cint`
`function FpStat (const path: RawByteString; var buf: Stat) : cint`
`function FpStat (path: ShortString; var buf: Stat) : cint`

Visibility: default

Description: `FpFStat` gets information about the file specified in `Path`, and stores it in `Info`, which is of type `stat` (249). The function returns zero if the call was successful, a nonzero return value indicates failure.

Errors: Extended error information can be retrieved using `fpGetErrno` (207).

`sys_enoent` `Path` does not exist.

See also: `FpStat` (237), `FpLStat` (214)

Listing: `./examples/bunixex/ex28.pp`

```

program example28;

{ Program to demonstrate the FStat function. }

uses BaseUnix;

var f : text;
    i : byte;
    info : stat;

begin
  { Make a file }
  assign (f,'test.fil');
  rewrite (f);
  for i:=1 to 10 do writeln (f,'Testline # ',i);
  close (f);
  { Do the call on made file. }
  if fpstat ('test.fil',info)<>0 then
    begin
      writeln('Fstat failed. Errno : ',fpgeterrno);
      halt (1);
    end;
  writeln;
  writeln ('Result of fstat on file "test.fil".');

```

```

writeln ('Inode : ',info.st_ino);
writeln ('Mode : ',info.st_mode);
writeln ('nlink : ',info.st_nlink);
writeln ('uid : ',info.st_uid);
writeln ('gid : ',info.st_gid);
writeln ('rdev : ',info.st_rdev);
writeln ('Size : ',info.st_size);
writeln ('Blksize : ',info.st_blksize);
writeln ('Blocks : ',info.st_blocks);
writeln ('atime : ',info.st_atime);
writeln ('mtime : ',info.st_mtime);
writeln ('ctime : ',info.st_ctime);
{ Remove file }
erase (f);
end.

```

1.4.83 fpSymlink

Synopsis: Create a symbolic link.

Declaration: `function fpSymlink(oldname: PChar; newname: PChar) : cint`

Visibility: default

Description: `SymLink` makes `NewName` point to the file in `OldName`, which doesn't necessarily exist. The two files DO NOT have the same inode number. This is known as a 'soft' link.

The permissions of the link are irrelevant, as they are not used when following the link. Ownership of the file is only checked in case of removal or renaming of the link.

The function returns zero if the call was successful, a nonzero value if the call failed.

Errors: Extended error information is returned by the `FpGetErrno` (207) function.

sys_eperm The file system containing `oldpath` and `newpath` does not support linking files.

sys_eaccess Write access for the directory containing `Newpath` is disallowed, or one of the directories in `OldPath` or `NewPath` has no search (=execute) permission.

sys_enoent A directory entry in `OldPath` or `NewPath` does not exist or is a symbolic link pointing to a non-existent directory.

sys_enotdir A directory entry in `OldPath` or `NewPath` is not a directory.

sys_enomem Insufficient kernel memory.

sys_erofs The files are on a read-only file system.

sys_eexist `NewPath` already exists.

sys_eloop `OldPath` or `NewPath` has a reference to a circular symbolic link, i.e. a symbolic link, whose expansion points to itself.

sys_enospc The device containing `NewPath` has no room for another entry.

See also: `FpLink` (212), `FpUnLink` (243), `FpReadLink` (225)

Listing: `./examples/unixex/ex22.pp`

Program Example22;

{ Program to demonstrate the SymLink and UnLink functions. }

Uses baseunix, Unix;

Var F : Text;
S : **String**;

```
begin
  Assign (F, 'test.txt');
  Rewrite (F);
  Writeln (F, 'This is written to test.txt');
  Close(f);
  { new.txt and test.txt are now the same file }
  if fpSymLink ('test.txt', 'new.txt') <> 0 then
    writeln ('Error when symlinking !');
  { Removing test.txt still leaves new.txt
    Pointing now to a non-existent file ! }
  If fpUnlink ('test.txt') <> 0 then
    Writeln ('Error when unlinking !');
  Assign (f, 'new.txt');
  { This should fail, since the symbolic link
    points to a non-existent file! }
  {$i-}
  Reset (F);
  {$i+}
  If IOResult=0 then
    Writeln ('This shouldn't happen');
  { Now remove new.txt also }
  If fpUnlink ('new.txt') <> 0 then
    Writeln ('Error when unlinking !');
end.
```

1.4.84 fpS_ISBLK

Synopsis: Is file a block device.

Declaration: function fpS_ISBLK(m: TMode) : Boolean

Visibility: default

Description: FpS_ISBLK checks the file mode m to see whether the file is a block device file. If so it returns True.

See also: FpFStat (204), FpS_ISLNK (240), FpS_ISREG (241), FpS_ISDIR (240), FpS_ISCHR (239), FpS_ISFIFO (240), FpS_ISSOCK (241)

1.4.85 fpS_ISCHR

Synopsis: Is file a character device.

Declaration: function fpS_ISCHR(m: TMode) : Boolean

Visibility: default

Description: FpS_ISCHR checks the file mode m to see whether the file is a character device file. If so it returns True.

See also: FpFStat (204), FpS_ISLNK (240), FpS_ISREG (241), FpS_ISDIR (240), FpS_ISBLK (239), FpS_ISFIFO (240), FpS_ISSOCK (241)

1.4.86 fpS_ISDIR

Synopsis: Is file a directory.

Declaration: `function fpS_ISDIR(m: TMode) : Boolean`

Visibility: default

Description: `fpS_ISDIR` checks the file mode `m` to see whether the file is a directory. If so, it returns `True`

See also: `FpFStat` (204), `FpS_ISLNK` (240), `FpS_ISREG` (241), `FpS_ISCHR` (239), `FpS_ISBLK` (239), `fpS_ISFIFO` (240), `FpS_ISSOCK` (241)

1.4.87 fpS_ISFIFO

Synopsis: Is file a FIFO.

Declaration: `function fpS_ISFIFO(m: TMode) : Boolean`

Visibility: default

Description: `FpS_ISFIFO` checks the file mode `m` to see whether the file is a fifo (a named pipe). If so it returns `True`.

See also: `FpFStat` (204), `FpS_ISLNK` (240), `FpS_ISREG` (241), `FpS_ISCHR` (239), `FpS_ISBLK` (239), `FpS_ISDIR` (240), `FpS_ISSOCK` (241)

1.4.88 fpS_ISLNK

Synopsis: Is file a symbolic link.

Declaration: `function fpS_ISLNK(m: TMode) : Boolean`

Visibility: default

Description: `FpS_ISLNK` checks the file mode `m` to see whether the file is a symbolic link. If so it returns `True`

See also: `FpFStat` (204), `FpS_ISFIFO` (240), `FpS_ISREG` (241), `FpS_ISCHR` (239), `FpS_ISBLK` (239), `FpS_ISDIR` (240), `FpS_ISSOCK` (241)

Listing: `./examples/bunixex/ex53.pp`

Program Example53;

{ Program to demonstrate the S_ISLNK function. }

Uses BaseUnix, Unix;

Var Info : Stat;

begin

```

  if fpLStat (paramstr(1), @info) = 0 then
    begin
      if fpS_ISLNK(info.st_mode) then
        Writeln ('File is a link');
      if fpS_ISREG(info.st_mode) then
        Writeln ('File is a regular file');
      if fpS_ISDIR(info.st_mode) then
        Writeln ('File is a directory');
```

```

    if fpS_ISCHR(info.st_mode) then
        Writeln ('File is a character device file');
    if fpS_ISBLK(info.st_mode) then
        Writeln ('File is a block device file');
    if fpS_ISFIFO(info.st_mode) then
        Writeln ('File is a named pipe (FIFO)');
    if fpS_ISSOCK(info.st_mode) then
        Writeln ('File is a socket');
    end;
end.

```

1.4.89 fpS_ISREG

Synopsis: Is file a regular file.

Declaration: `function fpS_ISREG(m: TMode) : Boolean`

Visibility: default

Description: `fpS_ISREG` checks the file mode `m` to see whether the file is a regular file. If so it returns `True`

See also: `FpFStat` (204), `fpS_ISFIFO` (240), `fpS_ISLNK` (240), `fpS_ISCHR` (239), `fpS_ISBLK` (239), `fpS_ISDIR` (240), `fpS_ISSOCK` (241)

1.4.90 fpS_ISSOCK

Synopsis: Is file a Unix socket.

Declaration: `function fpS_ISSOCK(m: TMode) : Boolean`

Visibility: default

Description: `fpS_ISSOCK` checks the file mode `m` to see whether the file is a socket. If so it returns `True`.

See also: `FpFStat` (204), `fpS_ISFIFO` (240), `fpS_ISLNK` (240), `fpS_ISCHR` (239), `fpS_ISBLK` (239), `fpS_ISDIR` (240), `fpS_ISREG` (241)

1.4.91 fptime

Synopsis: Return the current Unix time.

Declaration: `function FpTime(var tloc: TTime) : TTime`
`function fptime : time_t`

Visibility: default

Description: `FpTime` returns the number of seconds since 1970-01-01T00:00Z ignoring leap seconds. It is adjusted to the local time zone, but not to DST (daylight savings time). The result is also stored in `tloc` if it is specified.

Errors: On error, -1 is returned. Extended error information can be retrieved using `fpGetErrno` (207).

See also: `#rtl.dateutils.unixtodatetime` (695), `#rtl.dateutils.unixtimestampptomac` (694)

Listing: `./examples/bunixex/ex1.pp`

Program Example1;

{ Program to demonstrate the fptime function. }

Uses baseunix;

begin

Write ('Second past the start of the Epoch (1970-01-01T00:00): ');

Writeln (fptime);

end.

1.4.92 FpTimes

Synopsis: Return execution times for the current process.

Declaration: `function FpTimes(var buffer: tms) : TClock`

Visibility: default

Description: `fpTimes` stores the execution time of the current process and child processes in `buffer`.

The return value (on Linux) is the number of clock ticks since boot time. On error, -1 is returned, and extended error information can be retrieved with `fpGetErrno` ([207](#)).

See also: `fpUtime` ([243](#))

1.4.93 FpUmask

Synopsis: Set file creation mask.

Declaration: `function FpUmask(cmask: TMode) : TMode`

Visibility: default

Description: `fpUmask` changes the file creation mask for the current user to `cmask`. The current mask is returned.

See also: `fpChmod` ([195](#))

Listing: `./examples/bunixex/ex27.pp`

Program Example27;

{ Program to demonstrate the Umask function. }

Uses BaseUnix;

begin

Writeln ('Old Umask was : ',fpUmask(&111));

Writeln ('New Umask is : ',&111);

end.

1.4.94 FpUname

Synopsis: Return system name.

Declaration: `function FpUname(var name: UtsName) : cint`

Visibility: default

Description: `Uname` gets the name and configuration of the current Linux kernel, and returns it in the `name` record.

On success, 0 is returned, on error, -1 is returned.

Errors: Extended error information can be retrieved using `fpGetErrno` (207).

See also: `FpUTime` (243)

1.4.95 FpUnlink

Synopsis: Unlink (i.e. remove) a file.

Declaration: `function FpUnlink(path: PChar) : cint`
`function FpUnlink(const path: RawByteString) : cint`

Visibility: default

Description: `FpUnlink` decreases the link count on file `Path`. `Path` can be of type `AnsiString` or `PChar`. If the link count is zero, the file is removed from the disk.

The function returns zero if the call was successful, a nonzero value indicates failure.

Note: There exist a portable alternative to erase files: `system.erase`. Please use `fpUnlink` only if you are writing Unix specific code. `System.erase` will work on all operating systems.

For an example, see `FpLink` (212).

Errors: Extended error information can be retrieved using `fpGetErrno` (207).

sys_eaccess You have no write access right in the directory containing `Path`, or you have no search permission in one of the directory components of `Path`.

sys_eperm The directory containing pathname has the sticky-bit set and the process's effective uid is neither the uid of the file to be deleted nor that of the directory containing it.

sys_enoent A component of the path doesn't exist.

sys_enotdir A directory component of the path is not a directory.

sys_eisdir `Path` refers to a directory.

sys_enomem Insufficient kernel memory.

sys_erofs `Path` is on a read-only file system.

See also: `FpLink` (212), `FpSymLink` (238)

1.4.96 FpUtime

Synopsis: Set access and modification times of a file (touch).

Declaration: `function FpUtime(path: PChar; times: pUtimBuf) : cint`
`function FpUtime(const path: RawByteString; times: pUtimBuf) : cint`

Visibility: default

Description: `FpUtime` sets the access and modification times of the file specified in `Path`. the `times` record contains 2 fields, `actime`, and `modtime`, both of type `time_t` (commonly a `longint`). They should be filled with an epoch-like time, specifying, respectively, the last access time, and the last modification time. For some file systems (most notably, FAT), these times are the same.

The function returns zero on success, a nonzero return value indicates failure.

Errors: Extended error information can be retrieved using `fpGetErrno` (207).

sys_eaccess One of the directories in `Path` has no search (=execute) permission.

sys_enoent A directory entry in `Path` does not exist or is a symbolic link pointing to a non-existent directory.

Other errors may occur, but aren't documented.

See also: `FpTime` (241), `FpChown` (196), `FpAccess` (193)

Listing: `./examples/bunixex/ex25.pp`

Program Example25;

{ Program to demonstrate the UTime function. }

Uses Dos,BaseUnix,Unix,UnixUtil;

Var `utim` : `utimbuf`;
 `dow,msec,year,month,day,hour,minute,second` : `Word`;

```
begin
  { Set access and modification time of executable source }
  GetTime (hour,minute,second,msec);
  GetDate (year,month,day,dow);
  utim.actime:=LocalToEpoch(year,month,day,hour,minute,second);
  utim.modtime:=utim.actime;
  if Fputime('ex25.pp',@utim)<>0 then
    writeln ('Call to UTime failed !')
  else
    begin
      Write ('Set access and modification times to : ');
      Write (Hour:2,'.',minute:2,'.',second,' ');
      Write (Day:2,'/',month:2,'/',year:4);
      end;
    end.
```

1.4.97 FpWait

Synopsis: Wait for a child to exit.

Declaration: `function FpWait (var stat_loc: cint) : TPid`

Visibility: default

Description: `fpWait` suspends the current process and waits for any child to exit or stop due to a signal. It reports the exit status of the exited child in `stat_loc`.

The return value of the function is the process ID of the child that exited, or -1 on error.

Errors: Extended error information can be retrieved using `fpgetErrno` (207).

See also: `fpFork` (204), `fpExecve` (201), `fpWaitPid` (245)

1.4.98 FpWaitPid

Synopsis: Wait for a process to terminate.

Declaration: `function FpWaitpid(pid: TPid; stat_loc: pcint; options: cint) : TPid`
`function FpWaitPid(pid: TPid; var Status: cint; Options: cint) : TPid`

Visibility: default

Description: `fpWaitPid` waits for a child process with process ID `Pid` to exit. The value of `Pid` can be one of the following:

Pid < -1 Causes `fpWaitPid` to wait for any child process whose process group ID equals the absolute value of `pid`.

Pid = -1 Causes `fpWaitPid` to wait for any child process.

Pid = 0 Causes `fpWaitPid` to wait for any child process whose process group ID equals the one of the calling process.

Pid > 0 Causes `fpWaitPid` to wait for the child whose process ID equals the value of `Pid`.

The `Options` parameter can be used to specify further how `fpWaitPid` behaves:

WNOHANG Causes `fpWaitpid` to return immediately if no child has exited.

WUNTRACED Causes `fpWaitPid` to return also for children which are stopped, but whose status has not yet been reported.

__WCLONE Causes `fpWaitPid` also to wait for threads created by the `#rtl.linux.Clone` (1015) call.

The exit status of the process that caused `fpWaitPID` is reported in `stat_loc` or `Status`.

Upon return, it returns the process id of the process that exited, 0 if no process exited, or -1 in case of failure.

For an example, see `fpFork` (204).

Errors: Extended error information can be retrieved using `fpgetErrno` (207).

See also: `fpFork` (204), `fpExecve` (201), `fpWait` (244)

1.4.99 FpWrite

Synopsis: Write data to file descriptor.

Declaration: `function FpWrite(fd: cint; buf: PChar; nbytes: TSize) : TsSize`
`function FpWrite(fd: cint; const buf; nbytes: TSize) : TsSize`

Visibility: default

Description: `FpWrite` writes at most `nbytes` bytes from `buf` to file descriptor `fd`.

The function returns the number of bytes actually written, or -1 if an error occurred.

Errors: Extended error information can be retrieved using `fpGetErrno` (207).

See also: `FpOpen` (220), `FpClose` (197), `FpRead` (224), `FpFTruncate` (205), `FpLSeek` (213)

1.4.100 FpWriteV

Synopsis: Vector write: Write from multiple buffers to a file descriptor.

Declaration: `function FpWriteV(fd: cint; const iov: piovec; iovcnt: cint) : TsSize`

Visibility: default

Description: `FpWriteV` writes data to file descriptor `fd`. The data is taken from `iovcnt` buffers described by the `tiovec` (191) buffers pointed to by `iov`. It works like `fpWrite` (245) only from multiple buffers.

Errors: On error, -1 is returned.

See also: `FpReadV` (227), `FpPWrite` (224), `FpPRead` (223)

1.4.101 FreeShellArgV

Synopsis: Free the result of a `CreateShellArgV` (193) function.

Declaration: `procedure FreeShellArgV(p: PPChar)`

Visibility: default

Description: `FreeShellArgV` frees the memory pointed to by `P`, which was allocated by a call to `CreateShellArgV` (193).

Errors: None.

See also: `CreateShellArgV` (193)

1.4.102 wexitStatus

Synopsis: Extract the exit status from the `fpWaitPID` (245) result.

Declaration: `function wexitStatus(Status: cint) : cint`

Visibility: default

Description: `WEXITSTATUS` can be used to extract the exit status from `Status`, the result of the `FpWaitPID` (245) call.

See also: `FpWaitPID` (245), `WTERMSIG` (247), `WSTOPSIG` (247), `WIFEXITED` (246), `WIFSIGNALED` (247)

1.4.103 wifexited

Synopsis: Check whether the process exited normally.

Declaration: `function wifexited(Status: cint) : Boolean`

Visibility: default

Description: `WIFEXITED` checks `Status` and returns `True` if the status indicates that the process terminated normally, i.e. was not stopped by a signal.

See also: `FpWaitPID` (245), `WTERMSIG` (247), `WSTOPSIG` (247), `WIFSIGNALED` (247), `WEXITSTATUS` (246)

1.4.104 wifsignaled

Synopsis: Check whether the process was exited by a signal.

Declaration: `function wifsignaled(Status: cint) : Boolean`

Visibility: default

Description: `WIFSIGNALED` returns `True` if `Status` indicates that the process exited because it received a signal.

See also: `FpWaitPID` (245), `WTERMSIG` (247), `WSTOPSIG` (247), `WIFEXITED` (246), `WEXITSTATUS` (246)

1.4.105 wstopsig

Synopsis: Return the exit code from the process.

Declaration: `function wstopsig(Status: cint) : cint`

Visibility: default

Description: `WSTOPSIG` is an alias for `WEXITSTATUS` (246).

See also: `FpWaitPID` (245), `WTERMSIG` (247), `WIFEXITED` (246), `WIFSIGNALED` (247), `WEXITSTATUS` (246)

1.4.106 wtermSIG

Synopsis: Return the signal that caused a process to exit.

Declaration: `function wtermSIG(Status: cint) : cint`

Visibility: default

Description: `WTERMSIG` extracts from `Status` the signal number which caused the process to exit.

See also: `FpWaitPID` (245), `WSTOPSIG` (247), `WIFEXITED` (246), `WIFSIGNALED` (247), `WEXITSTATUS` (246)

1.5 Dir

```

Dir = record
  dd_fd : LongInt;
  dd_loc : LongInt;
  dd_size : LongInt
  ;
  dd_buf : pDirent;
  dd_nextoff : Cardinal;
  dd_max : Integer
  ;
  dd_lock : pointer;
end

```

Record used in `fpOpenDir` (221) and `fpReadDir` (225) calls.

1.6 Dirent

```
Dirent = record
  d_fileno : ino64_t;
  d_off : off_t;
  d_reclen
    : cushort;
  d_type : cuchar;
  d_name : Array[0..4095-sizeof(ino64_t
    )-sizeof(off_t)-sizeof(cushort)-sizeof(cuchar)] of Char;
end
```

Record used in the `fpReadDir` ([225](#)) function to return files in a directory.

1.7 FLock

```
FLock = record
  l_type : cushort;
  l_whence : cushort;
  l_start
    : kernel_off_t;
  l_len : kernel_off_t;
  l_pid : pid_t;
end
```

Lock description type for `fpFCntl` ([202](#)) lock call.

1.8 iovec

```
iovec = record
  iov_base : pointer;
  iov_len : size_t;
end
```

`iovec` is used in `freadv` ([227](#)) for IO to multiple buffers to describe a buffer location.

1.9 pollfd

```
pollfd = record
  fd : cint;
  events : cushort;
  revents : cushort
;
end
```

`pollfd` is used in the `fpPoll` ([223](#)) call to describe the various actions.

1.10 sigactionrec

```
sigactionrec = record
  sa_handler : sigactionhandler_t;
  sa_flags
    : culong;
  sa_restorer : sigrestorerhandler_t;
  sa_mask : sigset_t
;
end
```

Record used in `fpSigAction` ([232](#)) call.

1.11 Stat

```
Stat = packed record
  st_dev : QWord;
  __pad0_ : Array[0..3] of
    Byte;
  __st_ino_ : Cardinal;
  st_mode : Cardinal;
  st_nlink :
    Cardinal;
  st_uid : Cardinal;
  st_gid : Cardinal;
  st_rdev : QWord
;
  __pad3_ : Array[0..3] of Byte;
  st_size : QWord;
  st_blksize
    : Cardinal;
  st_blocks : QWord;
  st_atime : Cardinal;
  st_atime_nsec
    : Cardinal;
  st_mtime : Cardinal;
  st_mtime_nsec : Cardinal;
  st_ctime : Cardinal;
  st_ctime_nsec : Cardinal;
  st_ino : QWord
;
end
```

Record describing an inode (file) in the `FPFstat` ([204](#)) call.

1.12 tfpreg

```
tfpreg = record
  significand : Array[0..3] of Word;
  exponent :
    Word;
```

end

Record describing floating point register in signal handler.

1.13 tfpstate

```
tfpstate = record
  cw : Cardinal;
  sw : Cardinal;
  tag : Cardinal
;
  ipoff : Cardinal;
  cssel : Cardinal;
  dataoff : Cardinal;
  datasel : Cardinal;
  st : Array[0..7] of tfpreg;
  status : Word
;
  magic : Word;
  fxsr_env : Array[0..5] of DWord;
  mxcsr : DWord
;
  reserved : DWord;
  fxsr_st : Array[0..7] of tfpxreg;
  xmmreg
    : Array[0..7] of txmmreg;
case Byte of
1: (
  padding : Array[0.
    .43] of DWord;
case Byte of
1: (
  padding2 : Array[0..11] of DWord
  ;
);
2: (
  sw_reserved : tfpx_sw_bytes;
);
2: (
  padding1 : Array
    [0..43] of DWord;
);
end
```

Record describing floating point unit in signal handler.

1.14 tfpxreg

```
tfpxreg = record
  significand : Array[0..3] of Word;
```

```

    exponent
    : Word;
    padding : Array[0..2] of Word;
end

```

Record describing 16-byte floating point register in signal handler.

1.15 tfpx_sw_bytes

```

tfpx_sw_bytes = record
    magic1 : DWord;
    extended_size : DWord;
    xfeatures : QWord;
    xstate_size : DWord;
    padding : Array[0.
        .6] of DWord;
end

```

Extended state information in signal handler.

1.16 timezone

```

timezone = record
    tz_minuteswest : cint;
    tz_dsttime : cint;
end

```

Record describing a timezone.

1.17 tms

```

tms = record
    tms_utime : clock_t;
    tms_stime : clock_t;
    tms_cutime
    : clock_t;
    tms_cstime : clock_t;
end

```

Record containing timings for fpTimes ([242](#)) call.

1.18 TRLimit

```

TRLimit = record
    rlim_cur : rlim_t;
    rlim_max : rlim_t;
end

```

TRLimit is the structure used by the kernel to return resource limit information in.

1.19 tsigaltstack

```
tsigaltstack = record
  ss_sp : pointer;
  ss_flags : LongInt;
  ss_size
    : LongInt;
end
```

Provide the location of an alternate signal handler stack.

1.20 TSigContext

```
TSigContext = record
  gs : Word;
  __gsh : Word;
  fs : Word;
  __fsh
    : Word;
  es : Word;
  __esh : Word;
  ds : Word;
  __dsh : Word
  ;
  edi : Cardinal;
  esi : Cardinal;
  ebp : Cardinal;
  esp : Cardinal
  ;
  ebx : Cardinal;
  edx : Cardinal;
  ecx : Cardinal;
  eax : Cardinal
  ;
  trapno : Cardinal;
  err : Cardinal;
  eip : Cardinal;
  cs :
    Word;
  __csh : Word;
  eflags : Cardinal;
  esp_at_signal : Cardinal
  ;
  ss : Word;
  __ssh : Word;
  fpstate : pfpstate;
  oldmask : Cardinal
  ;
  cr2 : Cardinal;
```

end

This type is CPU dependent. Cross-platform code should not use the contents of this record.

1.21 tsiginfo

```
tsiginfo = record
  si_signo : LongInt;
  si_errno : LongInt;
  si_code
  : LongInt;
  _sifields : record
  case LongInt of
    0: (
      _pad
      : Array[0..(SI_PAD_SIZE)-1] of LongInt;
    );
    1: (
      _kill : record
      _pid : pid_t;
      _uid : uid_t;
      end;
    );
    2: (
      _timer : record
      _timer1 : DWord;
      _timer2 : DWord;
      end;
    );
    3: (
      _rt : record
      _pid : pid_t;
      _uid
      : uid_t;
      _sigval : pointer;
      end;
    );
    4: (
      _sigchld
      : record
      _pid : pid_t;
      _uid : uid_t;
      _status :
      LongInt;
      _utime : clock_t;
      _stime : clock_t;
      end
    );
    5: (
      _sigfault : record
      _addr : pointer;
```

```

        end
    ;
);
6: (
    _sigpoll : record
        _band : LongInt;
        _fd
    : LongInt;
    end;
);
end;
end

```

This type describes the signal that occurred.

1.22 TUcontext

```

TUcontext = record
    uc_flags : Cardinal;
    uc_link : Pucontext;
    uc_stack : tsigaltstack;
    uc_mcontext : TSigContext;
    uc_sigmask
    : tsigset;
end

```

This structure is used to describe the user context in a program or thread. It is not used in this unit, but is provided for completeness.

1.23 txmmreg

```

txmmreg = record
    element : Array[0..3] of DWord;
end

```

Record describing 16-byte MMX register in signal handler.

1.24 UTimBuf

```

UTimBuf = record
    actime : time_t;
    modtime : time_t;
end

```

Record used in `fpUtime` ([243](#)) to set file access and modification times.

1.25 UtsName

```
UtsName = record
  Sysname : Array[0..UTSNAME_LENGTH-1] of Char;
  Nodename : Array[0..UTSNAME_NODENAME_LENGTH-1] of Char;
  Release
    : Array[0..UTSNAME_LENGTH-1] of Char;
  Version : Array[0..UTSNAME_LENGTH
    -1] of Char;
  Machine : Array[0..UTSNAME_LENGTH-1] of Char;
  Domain
    : Array[0..UTSNAME_DOMAIN_LENGTH-1] of Charplatform;
end
```

The elements of this record are null-terminated C style strings, you cannot access them directly. Note that the `Domain` field is a GNU extension, and may not be available on all platforms.

Chapter 2

Reference for unit 'Character'

2.1 Used units

Table 2.1: Used units by unit 'Character'

Name	Page
System	1335
unicodedata	2150

2.2 Overview

The character unit contains the TCharacter ([264](#)) class, which consists mainly of class functions. It should not be constructed, but its class methods can be used. All class methods also exist as regular methods.

Many routines depend on Unicode collation data to be present in the binary (or distributed on disc with the application. This data can be loaded using the routines in the unicodedata ([2150](#)) unit.. The FPC project distributes some Unicode collation data in .bco files which can be loaded using the LoadCollation ([2161](#)) routine from that unit.

2.3 Constants, types and variables

2.3.1 Types

TCharacterOption = (coIgnoreInvalidSequence)

Table 2.2: Enumeration values for type TCharacterOption

Value	Explanation
coIgnoreInvalidSequence	Ignore invalid unicodecode sequences.

TCharacterOption is used in the toUpper ([264](#)) and toLower ([263](#)) functions to control the behaviour of the function.

TCharacterOptions = Set of TCharacterOption

TCharacterOptions is the set of TCharacterOption, used in toUpper (264) and toLower (263) functions to control the behaviour of the function.

```
TUnicodeCategory = (ucUppercaseLetter, ucLowercaseLetter,  
    ucTitlecaseLetter, ucModifierLetter, ucOtherLetter,  
    ucNonSpacingMark, ucCombiningMark, ucEnclosingMark  
,  
    ucDecimalNumber, ucLetterNumber, ucOtherNumber  
,  
    ucConnectPunctuation, ucDashPunctuation,  
    ucOpenPunctuation, ucClosePunctuation,  
    ucInitialPunctuation, ucFinalPunctuation,  
    ucOtherPunctuation, ucMathSymbol, ucCurrencySymbol,  
    ucModifierSymbol, ucOtherSymbol, ucSpaceSeparator,  
    ucLineSeparator, ucParagraphSeparator, ucControl,  
    ucFormat, ucSurrogate, ucPrivateUse, ucUnassigned)
```

Table 2.3: Enumeration values for type TUnicodeCategory

Value	Explanation
ucClosePunctuation	Punctuation, close (Pe).
ucCombiningMark	Mark, spacing combining (Mc).
ucConnectPunctuation	Punctuation, connector (Pc).
ucControl	Other, control (Cc).
ucCurrencySymbol	Symbol, currency (Sc).
ucDashPunctuation	Punctuation, dash (Pd).
ucDecimalNumber	Number, decimal digit (Nd).
ucEnclosingMark	Mark, enclosing (Me).
ucFinalPunctuation	Punctuation, final quote (Pf, may behave like Ps or Pe depending on usage).
ucFormat	Other, format (Cf).
ucInitialPunctuation	Punctuation, initial quote (Pi, may behave like Ps or Pe depending on usage).
ucLetterNumber	Number, letter (Nl).
ucLineSeparator	Separator, line (Zl).
ucLowercaseLetter	Letter, lowercase (Ll).
ucMathSymbol	Symbol, math (Sm).
ucModifierLetter	Letter, modifier (Lm).
ucModifierSymbol	Symbol, modifier (Sk).
ucNonSpacingMark	Mark, nonspacing (Mn).
ucOpenPunctuation	Punctuation, open (Ps).
ucOtherLetter	Letter, other (Lo).
ucOtherNumber	Number, other (No).
ucOtherPunctuation	Punctuation, other (Po).
ucOtherSymbol	Symbol, other (So).
ucParagraphSeparator	Separator, paragraph (Zp).
ucPrivateUse	Other, private use (Co).
ucSpaceSeparator	Separator, space (Zs).
ucSurrogate	Other, surrogate (Cs).
ucTitlecaseLetter	Letter, titlecase (Lt).
ucUnassigned	Other, not assigned (including noncharacters) (Cn).
ucUppercaseLetter	Letter, uppercase (Lu).

This enumeration type contains the characterization of all possible Unicode characters. It is used in the `GetUnicodeCategory` (259) and `TCharacter.GetUnicodeCategory` (266) functions.

`TUnicodeCategorySet = Set of TUnicodeCategory`

`TUnicodeCategorySet` is the set of `TUnicodeCategory` (257). It is used internally in the `TCharacter` (264) class.

2.4 Procedures and functions

2.4.1 ConvertFromUtf32

Synopsis: alias for `TCharacter.ConvertFromUtf32`.

Declaration: `function ConvertFromUtf32 (AChar: UCS4Char) : UnicodeString`

Visibility: default

Description: `ConvertFromUtf32` is a shortcut for `TCharacter.ConvertFromUtf32` (265).

See also: `TCharacter.ConvertFromUtf32` (265)

2.4.2 ConvertToUtf32

Synopsis: alias for `TCharacter.ConvertToUtf32`.

Declaration:

```
function ConvertToUtf32(const AString: UnicodeString; AIndex: Integer)
                        : UCS4Char; Overload
function ConvertToUtf32(const AString: UnicodeString; AIndex: Integer;
                        out ACharLength: Integer) : UCS4Char; Overload
function ConvertToUtf32(const AHighSurrogate: UnicodeChar;
                        const ALowSurrogate: UnicodeChar) : UCS4Char
                        ; Overload
```

Visibility: default

Description: `ConvertToUtf32` is a shortcut for `TCharacter.ConvertToUtf32` (265).

See also: `TCharacter.ConvertToUtf32` (265)

2.4.3 GetNumericValue

Synopsis: Alias for `TCharacter.GetNumericValue`.

Declaration:

```
function GetNumericValue(AChar: UnicodeChar) : Double; Overload
function GetNumericValue(const AString: UnicodeString; AIndex: Integer)
                        : Double; Overload
```

Visibility: default

Description: `GetNumericValue` is a shortcut for `TCharacter.GetNumericValue` (266).

See also: `TCharacter.GetNumericValue` (266)

2.4.4 GetUnicodeCategory

Synopsis: Alias for `TCharacter.GetUnicodeCategory`.

Declaration:

```
function GetUnicodeCategory(AChar: UnicodeChar) : TUnicodeCategory
                        ; Overload
function GetUnicodeCategory(const AString: UnicodeString;
                        AIndex: Integer) : TUnicodeCategory
                        ; Overload
```

Visibility: default

Description: `GetUnicodeCategory` is a shortcut for `TCharacter.GetUnicodeCategory` (266).

See also: `TCharacter.GetUnicodeCategory` (266)

2.4.5 IsControl

Synopsis: Alias for `TCharacter.IsControl`.

Declaration: `function IsControl(AChar: UnicodeChar) : Boolean; Overload`
`function IsControl(const AString: UnicodeString; AIndex: Integer)`
`: Boolean; Overload`

Visibility: default

Description: `IsControl` is a shortcut for `TCharacter.IsControl` (266).

See also: `TCharacter.IsControl` (266)

2.4.6 IsDigit

Synopsis: Alias for `TCharacter.IsDigit`.

Declaration: `function IsDigit(AChar: UnicodeChar) : Boolean; Overload`
`function IsDigit(const AString: UnicodeString; AIndex: Integer)`
`: Boolean; Overload`

Visibility: default

Description: `IsDigit` is a shortcut for `TCharacter.IsDigit` (267).

See also: `TCharacter.IsDigit` (267)

2.4.7 IsHighSurrogate

Synopsis: Alias for `TCharacter.IsHighSurrogate`.

Declaration: `function IsHighSurrogate(AChar: UnicodeChar) : Boolean; Overload`
`function IsHighSurrogate(const AString: UnicodeString; AIndex: Integer)`
`: Boolean; Overload`

Visibility: default

Description: `IsHighSurrogate` is a shortcut for `TCharacter.IsHighSurrogate` (268)

See also: `TCharacter.IsHighSurrogate` (268)

2.4.8 IsLetter

Synopsis: Alias for `TCharacter.IsLetter`.

Declaration: `function IsLetter(AChar: UnicodeChar) : Boolean; Overload`
`function IsLetter(const AString: UnicodeString; AIndex: Integer)`
`: Boolean; Overload`

Visibility: default

Description: `IsLetter` is a shortcut for `TCharacter.IsLetter` (269)

See also: `TCharacter.IsLetter` (269)

2.4.9 IsLetterOrDigit

Synopsis: Alias for `TCharacter.IsLetterOrDigit`.

Declaration: `function IsLetterOrDigit (AChar: UnicodeChar) : Boolean; Overload`
`function IsLetterOrDigit (const AString: UnicodeString; AIndex: Integer)`
`: Boolean; Overload`

Visibility: default

Description: `IsLetterOrDigit` is a shortcut for `TCharacter.IsLetterOrDigit` ([269](#)).

See also: `TCharacter.IsLetterOrDigit` ([269](#))

2.4.10 IsLower

Synopsis: Alias for `TCharacter.IsLower`.

Declaration: `function IsLower (AChar: UnicodeChar) : Boolean; Overload`
`function IsLower (const AString: UnicodeString; AIndex: Integer)`
`: Boolean; Overload`

Visibility: default

Description: `IsLower` is a shortcut for `TCharacter.IsLower` ([270](#))

See also: `TCharacter.IsLower` ([270](#))

2.4.11 IsLowSurrogate

Synopsis: Alias for `TCharacter.IsLowSurrogate`.

Declaration: `function IsLowSurrogate (AChar: UnicodeChar) : Boolean; Overload`
`function IsLowSurrogate (const AString: UnicodeString; AIndex: Integer)`
`: Boolean; Overload`

Visibility: default

Description: `IsLowSurrogate` is a shortcut for `TCharacter.IsLowSurrogate` ([268](#))

See also: `TCharacter.IsLowSurrogate` ([268](#))

2.4.12 IsNumber

Synopsis: Alias for `TCharacter.IsNumber`.

Declaration: `function IsNumber (AChar: UnicodeChar) : Boolean; Overload`
`function IsNumber (const AString: UnicodeString; AIndex: Integer)`
`: Boolean; Overload`

Visibility: default

Description: `IsNumber` is a shortcut for `TCharacter.IsNumber` ([270](#))

See also: `TCharacter.IsNumber` ([270](#))

2.4.13 IsPunctuation

Synopsis: Alias for `TCharacter.IsPunctuation`.

Declaration: `function IsPunctuation(AChar: UnicodeChar) : Boolean; Overload`
`function IsPunctuation(const AString: UnicodeString; AIndex: Integer)`
`: Boolean; Overload`

Visibility: default

Description: `IsPunctuation` is a shortcut for `TCharacter.IsPunctuation` ([271](#))

See also: `TCharacter.IsPunctuation` ([271](#))

2.4.14 IsSeparator

Synopsis: Alias for `TCharacter.IsSeparator`.

Declaration: `function IsSeparator(AChar: UnicodeChar) : Boolean; Overload`
`function IsSeparator(const AString: UnicodeString; AIndex: Integer)`
`: Boolean; Overload`

Visibility: default

Description: `IsSeparator` is a shortcut for `TCharacter.IsSeparator` ([271](#))

See also: `TCharacter.IsSeparator` ([271](#))

2.4.15 IsSurrogate

Synopsis: Alias for `TCharacter.IsSurrogate`.

Declaration: `function IsSurrogate(AChar: UnicodeChar) : Boolean; Overload`
`function IsSurrogate(const AString: UnicodeString; AIndex: Integer)`
`: Boolean; Overload`

Visibility: default

Description: `IsSurrogate` is a shortcut for `TCharacter.IsSurrogate` ([267](#)).

See also: `TCharacter.IsSurrogate` ([267](#))

2.4.16 IsSurrogatePair

Synopsis: Alias for `TCharacter.IsSurrogatePair`.

Declaration: `function IsSurrogatePair(const AHighSurrogate: UnicodeChar;`
`const ALowSurrogate: UnicodeChar) : Boolean`
`; Overload`
`function IsSurrogatePair(const AString: UnicodeString; AIndex: Integer)`
`: Boolean; Overload`

Visibility: default

Description: `IsSurrogatePair` is a shortcut for `TCharacter.IsSurrogatePair` ([269](#))

See also: `TCharacter.IsSurrogatePair` ([269](#))

2.4.17 IsSymbol

Synopsis: Alias for `TCharacter.IsSymbol`.

Declaration: `function IsSymbol(AChar: UnicodeChar) : Boolean; Overload`
`function IsSymbol(const AString: UnicodeString; AIndex: Integer)`
`: Boolean; Overload`

Visibility: default

Description: `IsSymbol` is a shortcut for `TCharacter.IsSymbol` ([272](#))

See also: `TCharacter.IsSymbol` ([272](#))

2.4.18 IsUpper

Synopsis: Alias for `TCharacter.IsUpper`.

Declaration: `function IsUpper(AChar: UnicodeChar) : Boolean; Overload`
`function IsUpper(const AString: UnicodeString; AIndex: Integer)`
`: Boolean; Overload`

Visibility: default

Description: `IsUpper` is a shortcut for `TCharacter.IsUpper` ([272](#))

See also: `TCharacter.IsUpper` ([272](#))

2.4.19 IsWhiteSpace

Synopsis: Alias for `TCharacter.IsWhiteSpace`.

Declaration: `function IsWhiteSpace(AChar: UnicodeChar) : Boolean; Overload`
`function IsWhiteSpace(const AString: UnicodeString; AIndex: Integer)`
`: Boolean; Overload`

Visibility: default

Description: `IsWhiteSpace` is a shortcut for `TCharacter.IsWhiteSpace` ([272](#))

See also: `TCharacter.IsWhiteSpace` ([272](#))

2.4.20 ToLower

Synopsis: Alias for `TCharacter.ToLower`.

Declaration: `function ToLower(AChar: UnicodeChar) : UnicodeChar; Overload`
`function ToLower(const AString: UnicodeString) : UnicodeString`
`; Overload`

Visibility: default

Description: `ToLower` is a shortcut for `TCharacter.ToLower` ([273](#))

See also: `TCharacter.ToLower` ([273](#))

2.4.21 ToUpper

Synopsis: Alias for `TCharacter.ToUpper`.

Declaration: `function ToUpper(AChar: UnicodeChar) : UnicodeChar; Overload`
`function ToUpper(const AString: UnicodeString) : UnicodeString`
`; Overload`

Visibility: default

Description: `ToUpper` is a shortcut for `TCharacter.ToUpper` ([273](#))

See also: `TCharacter.ToUpper` ([273](#))

2.5 TCharacter

2.5.1 Description

`TCharacter` is provided for Delphi compatibility. All it's class functions and methods are also available as regular functions.

2.5.2 Method overview

Page	Method	Description
265	<code>ConvertFromUtf32</code>	Convert a UTF32 character to <code>UnicodeString</code> .
265	<code>ConvertToUtf32</code>	Convert a UTF16 character to a UTF32 character.
264	<code>Create</code>	Constructor (do not call).
266	<code>GetNumericValue</code>	Get the numeric value of the character.
266	<code>GetUnicodeCategory</code>	Get the Unicode category of a character.
266	<code>IsControl</code>	Check whether a Unicode character is a Unicode control character.
267	<code>IsDigit</code>	Check whether a Unicode character is a digit.
268	<code>IsHighSurrogate</code>	Check whether a Unicode character is a surrogate in the high range.
269	<code>IsLetter</code>	Check if a Unicode character is a letter.
269	<code>IsLetterOrDigit</code>	Check if a Unicode character is a letter or digit.
270	<code>IsLower</code>	Check if a Unicode character is a lowercase letter.
268	<code>IsLowSurrogate</code>	Check whether a Unicode character is a surrogate in the low range.
270	<code>IsNumber</code>	Check if a Unicode character is a number.
271	<code>IsPunctuation</code>	Check if a Unicode character is a punctuation character.
271	<code>IsSeparator</code>	Check if a Unicode character is a separator character.
267	<code>IsSurrogate</code>	Check whether a Unicode character is a surrogate.
269	<code>IsSurrogatePair</code>	Check if a pair of characters is a set of high/low surrogate characters.
272	<code>IsSymbol</code>	Check if a Unicode character is a symbol character.
272	<code>IsUpper</code>	Check whether a Unicode character is an uppercase letter.
272	<code>IsWhiteSpace</code>	Check whether a Unicode character is a whitespace character.
273	<code>ToLower</code>	Convert a character or string to lowercase.
273	<code>ToUpper</code>	Convert a character or string to uppercase.

2.5.3 TCharacter.Create

Synopsis: Constructor (do not call).

Declaration: `constructor Create`

Visibility: `public`

Description: `Create` is provided for completeness and Delphi compatibility, but should not be called in FPC code, it will raise an exception.

Errors: Any attempt to call `Create` will result in an exception being raised.

2.5.4 TCharacter.ConvertFromUtf32

Synopsis: Convert a UTF32 character to UnicodeString.

Declaration: `class function ConvertFromUtf32(AChar: UCS4Char) : UnicodeString; Static`

Visibility: `public`

Description: `TCharacter.ConvertFromUtf32` converts a single UTF32 character `AChar` to a UTF16 string. This is the opposite of `TCharacter.ConvertToUtf32` (265).

The result is a string, since multiple UTF16 characters can be needed to encode a single UTF32 character.

Errors: If `AChar` is not in the valid range of UTF32 characters, an `EArgumentOutOfRangeException` (256) exception is raised.

See also: `EArgumentOutOfRangeException` (256), `TCharacter.ConvertToUtf32` (265)

2.5.5 TCharacter.ConvertToUtf32

Synopsis: Convert a UTF16 character to a UTF32 character.

```
Declaration: class function ConvertToUtf32(const AString: UnicodeString;
                                           AIndex: Integer) : UCS4Char; Overload
; Static
class function ConvertToUtf32(const AString: UnicodeString;
                               AIndex: Integer; out ACharLength: Integer)
                               : UCS4Char; Overload; Static
class function ConvertToUtf32(const AHighSurrogate: UnicodeChar;
                               const ALowSurrogate: UnicodeChar)
                               : UCS4Char; Overload; Static
```

Visibility: `public`

Description: `TCharacter.ConvertToUtf32` converts a UTF16-encoded Unicode character to a Unicode32 character. This is the opposite of `TCharacter.ConvertFromUtf32` (265). The function exists in several overloaded versions, to be able to present the Unicode character in one of 2 ways:

1. As a position `AIndex` (in unicodechar units) in a string `AString` to a Unicode32 character. The source is a string, since multiple UTF16 characters can be needed to encode a single UTF32 character. In this form, Optionally, the character length (1 or 2) can be returned in `ACharLength`.
2. As 2 UTF16 Unicode characters, representing the high and low surrogate pairs: `AHighSurrogate` and `ALowSurrogate`.

Errors: If `AIndex` is not a valid character index in the string `AString`, an `EArgumentOutOfRangeException` (256) exception is raised. If the character at that position is not complete, an `EArgumentOutOfRangeException` (256) exception is raised.

See also: `TCharacter.ConvertFromUtf32` (265), `EArgumentOutOfRangeException` (256), `EArgumentOutOfRangeException` (256)

2.5.6 TCharacter.GetNumericValue

Synopsis: Get the numeric value of the character.

Declaration:

```
class function GetNumericValue(AChar: UnicodeChar) : Double; Overload
; Static
class function GetNumericValue(const AString: UnicodeString;
                               AIndex: Integer) : Double; Overload
; Static
```

Visibility: public

Description: `TCharacter.GetNumericValue` returns the numerical value (ID) of the Unicode character. The character can be presented in 2 ways: `AChar`, a UTF16 Unicode character, or a surrogate pair in a Unicode string `AString` starting at position `AIndex`.

Errors: If `AIndex` is not a valid character index in the string `AString`, an `EArgumentOutOfRangeException` (256) exception is raised. If the character at that position is not complete, an `EArgumentOutOfRangeException` (256) exception is raised.

See also: `TCharacter.GetUnicodeCategory` (266)

2.5.7 TCharacter.GetUnicodeCategory

Synopsis: Get the Unicode category of a character.

Declaration:

```
class function GetUnicodeCategory(AChar: UnicodeChar) : TUnicodeCategory
; Overload; Static
class function GetUnicodeCategory(const AString: UnicodeString;
                                   AIndex: Integer) : TUnicodeCategory
; Overload; Static
```

Visibility: public

Description: `TCharacter.GetUnicodeCategory` returns the Unicode category of a character. The character can be presented in 2 ways: `AChar`, a UTF16 Unicode character, or a surrogate pair in a Unicode string `AString` starting at position `AIndex`.

Errors: If `AIndex` is not a valid character index in the string `AString`, an `EArgumentOutOfRangeException` (256) exception is raised. If the character at that position is not complete, an `EArgumentOutOfRangeException` (256) exception is raised.

See also: `TUnicodeCategory` (257)

2.5.8 TCharacter.IsControl

Synopsis: Check whether a Unicode character is a Unicode control character.

Declaration: `class function IsControl(AChar: UnicodeChar) : Boolean; Overload
; Static
class function IsControl(const AString: UnicodeString; AIndex: Integer)
: Boolean; Overload; Static`

Visibility: public

Description: `IsControl` returns `True` if a Unicode character has category `ucControl`. The character can be specified as a UTF16 character `AChar` or a UTF16 encoded character starting at position `AIndex` in string `AString`.

Errors: If `AIndex` is not a valid character index in the string `AString`, an `EArgumentOutOfRangeException` (256) exception is raised. If the character at that position is not complete, an `EArgumentOutOfRangeException` (256) exception is raised.

See also: `GetUnicodeCategory` (266), `IsDigit` (267), `IsSurrogate` (267), `IsHighSurrogate` (268), `IsLowSurrogate` (268), `IsSurrogatePair` (269), `IsLetter` (269), `IsLetterOrDigit` (269), `IsLower` (270), `IsNumber` (270), `IsPunctuation` (271), `IsSeparator` (271), `IsSymbol` (272), `IsUpper` (272), `IsWhiteSpace` (272)

2.5.9 TCharacter.IsDigit

Synopsis: Check whether a Unicode character is a digit.

Declaration: `class function IsDigit(AChar: UnicodeChar) : Boolean; Overload; Static
class function IsDigit(const AString: UnicodeString; AIndex: Integer)
: Boolean; Overload; Static`

Visibility: public

Description: `IsDigit` returns `True` if a Unicode character has category `ucDecimalNumber`. The character can be specified as a UTF16 character `AChar` or a UTF16 encoded character starting at position `AIndex` in string `AString`.

Errors: If `AIndex` is not a valid character index in the string `AString`, an `EArgumentOutOfRangeException` (256) exception is raised. If the character at that position is not complete, an `EArgumentOutOfRangeException` (256) exception is raised.

See also: `IsControl` (266), `IsDigit` (267), `IsSurrogate` (267), `IsHighSurrogate` (268), `IsLowSurrogate` (268), `IsSurrogatePair` (269), `IsLetter` (269), `IsLetterOrDigit` (269), `IsLower` (270), `IsNumber` (270), `IsPunctuation` (271), `IsSeparator` (271), `IsSymbol` (272), `IsUpper` (272), `IsWhiteSpace` (272)

2.5.10 TCharacter.IsSurrogate

Synopsis: Check whether a Unicode character is a surrogate.

Declaration: `class function IsSurrogate(AChar: UnicodeChar) : Boolean; Overload
; Static
class function IsSurrogate(const AString: UnicodeString;
AIndex: Integer) : Boolean; Overload; Static`

Visibility: public

Description: `IsSurrogate` returns `True` if a Unicode character has category `ucSurrogate`. The character can be specified as a UTF16 character `AChar` or a UTF16 encoded character starting at position `AIndex` in string `AString`.

Errors: If `AIndex` is not a valid character index in the string `AString`, an `EArgumentOutOfRangeException` (256) exception is raised. If the character at that position is not complete, an `EArgumentException` (256) exception is raised.

See also: `EArgumentException` (256), `IsControl` (266), `IsDigit` (267), `IsHighSurrogate` (268), `IsLowSurrogate` (268), `IsLetter` (269), `IsLetterOrDigit` (269), `IsLower` (270), `IsNumber` (270), `IsPunctuation` (271), `IsSymbol` (272), `IsUpper` (272), `IsWhiteSpace` (272)

2.5.11 TCharacter.IsHighSurrogate

Synopsis: Check whether a Unicode character is a surrogate in the high range.

Declaration:

```
class function IsHighSurrogate(AChar: UnicodeChar) : Boolean; Overload
    ; Static
class function IsHighSurrogate(const AString: UnicodeString;
    AIndex: Integer) : Boolean; Overload
    ; Static
```

Visibility: public

Description: `IsHighSurrogate` returns `True` if a Unicode character has category `ucSurrogate` and is in the high range of the surrogate characters (between `HIGH_SURROGATE_BEGIN` and `HIGH_SURROGATE_END`). The character can be specified as a UTF16 character `AChar` or a UTF16 encoded character starting at position `AIndex` in string `AString`.

Errors: If `AIndex` is not a valid character index in the string `AString`, an `EArgumentOutOfRangeException` (256) exception is raised. If the character at that position is not complete, an `EArgumentException` (256) exception is raised.

See also: `EArgumentException` (256), `IsControl` (266), `IsDigit` (267), `IsSurrogate` (267), `IsLowSurrogate` (268), `IsLetter` (269), `IsLetterOrDigit` (269), `IsLower` (270), `IsNumber` (270), `IsPunctuation` (271), `IsSymbol` (272), `IsUpper` (272), `IsWhiteSpace` (272)

2.5.12 TCharacter.IsLowSurrogate

Synopsis: Check whether a Unicode character is a surrogate in the low range.

Declaration:

```
class function IsLowSurrogate(AChar: UnicodeChar) : Boolean; Overload
    ; Static
class function IsLowSurrogate(const AString: UnicodeString;
    AIndex: Integer) : Boolean; Overload
    ; Static
```

Visibility: public

Description: `IsLowSurrogate` returns `True` if a Unicode character has category `ucSurrogate` and is in the low range of the surrogate characters (between `LOW_SURROGATE_BEGIN` and `LOW_SURROGATE_END`). The character can be specified as a UTF16 character `AChar` or a UTF16 encoded character starting at position `AIndex` in string `AString`.

Errors: If `AIndex` is not a valid character index in the string `AString`, an `EArgumentOutOfRangeException` (256) exception is raised. If the character at that position is not complete, an `EArgumentException` (256) exception is raised.

See also: `EArgumentException` (256), `IsControl` (266), `IsDigit` (267), `IsSurrogate` (267), `IsHighSurrogate` (268), `IsLetter` (269), `IsLetterOrDigit` (269), `IsLower` (270), `IsNumber` (270), `IsPunctuation` (271), `IsSymbol` (272), `IsUpper` (272), `IsWhiteSpace` (272)

2.5.13 TCharacter.IsSurrogatePair

Synopsis: Check if a pair of characters is a set of high/low surrogate characters.

Declaration:

```
class function IsSurrogatePair(const AHighSurrogate: UnicodeChar;
                               const ALowSurrogate: UnicodeChar)
                               : Boolean; Overload; Static
class function IsSurrogatePair(const AString: UnicodeString;
                               AIndex: Integer) : Boolean; Overload
                               ; Static
```

Visibility: public

Description: `IsSurrogatePair` returns `True` if `AHighSurrogate` and `ALowSurrogate` form a valid Unicode surrogate pair. (`AHighSurrogate` is a high surrogate and `ALowSurrogate` a matching low surrogate) The character can be specified as a UTF16 character `AChar` or a pair of UTF16 encoded characters starting at position `AIndex` in string `AString`.

Errors: If `AIndex` is not a valid character index in the string `AString`, an `EArgumentOutOfRangeException` (256) exception is raised. If the character at that position is not complete, an `EArgumentOutOfRangeException` (256) exception is raised.

See also: `EArgumentOutOfRangeException` (256), `IsControl` (266), `IsDigit` (267), `IsSurrogate` (267), `IsHighSurrogate` (268), `IsLowSurrogate` (268), `IsLetter` (269), `IsLetterOrDigit` (269), `IsLower` (270), `IsNumber` (270), `IsPunctuation` (271), `IsSymbol` (272), `IsUpper` (272), `IsWhiteSpace` (272)

2.5.14 TCharacter.IsLetter

Synopsis: Check if a Unicode character is a letter.

Declaration:

```
class function IsLetter(AChar: UnicodeChar) : Boolean; Overload
                               ; Static
class function IsLetter(const AString: UnicodeString; AIndex: Integer)
                               : Boolean; Overload; Static
```

Visibility: public

Description: `IsLetter` returns `True` if a Unicode character has category that is one of the letter categories (`ucUppercaseLetter`, `ucLowercaseLetter`, `ucTitlecaseLetter`, `ucModifierLetter`, `ucOtherLetter`). The character can be specified as a UTF16 character `AChar` or a UTF16 encoded character starting at position `AIndex` in string `AString`.

Errors: If `AIndex` is not a valid character index in the string `AString`, an `EArgumentOutOfRangeException` (256) exception is raised. If the character at that position is not complete, an

See also: `EArgumentOutOfRangeException` (256), `IsControl` (266), `IsDigit` (267), `IsSurrogate` (267), `IsHighSurrogate` (268), `IsLowSurrogate` (268), `IsSurrogatePair` (269), `IsLetter` (269), `IsLetterOrDigit` (269), `IsLower` (270), `IsNumber` (270), `IsPunctuation` (271), `IsSymbol` (272), `IsUpper` (272), `IsWhiteSpace` (272)

2.5.15 TCharacter.IsLetterOrDigit

Synopsis: Check if a Unicode character is a letter or digit.

Declaration:

```
class function IsLetterOrDigit(AChar: UnicodeChar) : Boolean; Overload
                               ; Static
class function IsLetterOrDigit(const AString: UnicodeString;
                               AIndex: Integer) : Boolean; Overload
                               ; Static
```

Visibility: public

Description: `IsLetterOrDigit` returns `True` if a Unicode character has category that is one of the letter categories (`ucUppercaseLetter`, `ucLowercaseLetter`, `ucTitlecaseLetter`, `ucModifierLetter`, `ucOtherLetter`, `ucDecimalNumber`, `ucLetterNumber`). The character can be specified as a UTF16 character `AChar` or a UTF16 encoded character starting at position `AIndex` in string `AString`.

Errors: If `AIndex` is not a valid character index in the string `AString`, an `EArgumentOutOfRangeException` (256) exception is raised. If the character at that position is not complete, an `EArgumentOutOfRangeException` (256) exception is raised.

See also: `IsControl` (266), `IsDigit` (267), `IsSurrogate` (267), `IsHighSurrogate` (268), `IsLowSurrogate` (268), `IsSurrogatePair` (269), `IsLetter` (269), `IsLower` (270), `IsNumber` (270), `IsPunctuation` (271), `IsSymbol` (272), `IsUpper` (272), `IsWhiteSpace` (272)

2.5.16 TCharacter.IsLower

Synopsis: Check if a Unicode character is a lowercase letter.

Declaration: `class function IsLower(AChar: UnicodeChar) : Boolean; Overload; Static`
`class function IsLower(const AString: UnicodeString; AIndex: Integer)`
`: Boolean; Overload; Static`

Visibility: public

Description: `IsLower` returns `True` if a Unicode character has category `ucLowercaseLetter`. The character can be specified as a UTF16 character `AChar` or a UTF16 encoded character starting at position `AIndex` in string `AString`.

Errors: If `AIndex` is not a valid character index in the string `AString`, an `EArgumentOutOfRangeException` (256) exception is raised. If the character at that position is not complete, an `EArgumentOutOfRangeException` (256) exception is raised.

See also: `IsControl` (266), `IsDigit` (267), `IsSurrogate` (267), `IsHighSurrogate` (268), `IsLowSurrogate` (268), `IsSurrogatePair` (269), `IsLetter` (269), `IsLetterOrDigit` (269), `IsNumber` (270), `IsPunctuation` (271), `IsSymbol` (272), `IsUpper` (272), `IsWhiteSpace` (272)

2.5.17 TCharacter.IsNumber

Synopsis: Check if a Unicode character is a number.

Declaration: `class function IsNumber(AChar: UnicodeChar) : Boolean; Overload`
`; Static`
`class function IsNumber(const AString: UnicodeString; AIndex: Integer)`
`: Boolean; Overload; Static`

Visibility: public

Description: `IsNumber` returns `True` if a Unicode character has category that is one of the number categories (`ucDecimalNumber`, `ucLetterNumber`, `ucOtherNumber`). The character can be specified as a UTF16 character `AChar` or a UTF16 encoded character starting at position `AIndex` in string `AString`.

Errors: If `AIndex` is not a valid character index in the string `AString`, an `EArgumentOutOfRangeException` (256) exception is raised. If the character at that position is not complete, an `EArgumentOutOfRangeException` (256) exception is raised.

See also: [IsControl \(266\)](#), [IsDigit \(267\)](#), [IsSurrogate \(267\)](#), [IsHighSurrogate \(268\)](#), [IsLowSurrogate \(268\)](#), [IsSurrogatePair \(269\)](#), [IsLetter \(269\)](#), [IsLetterOrDigit \(269\)](#), [IsLower \(270\)](#), [IsNumber \(270\)](#), [IsPunctuation \(271\)](#), [IsSymbol \(272\)](#), [IsUpper \(272\)](#), [IsWhiteSpace \(272\)](#)

2.5.18 TCharacter.IsPunctuation

Synopsis: Check if a Unicode character is a punctuation character.

Declaration:

```
class function IsPunctuation(AChar: UnicodeChar) : Boolean; Overload
                        ; Static
class function IsPunctuation(const AString: UnicodeString;
                        AIndex: Integer) : Boolean; Overload
                        ; Static
```

Visibility: public

Description: `IsPunctuation` returns `True` if a Unicode character has category that is one of the punctuation categories (`ucConnectPunctuation`, `ucDashPunctuation`, `ucOpenPunctuation`, `ucClosePunctuation`, `ucInitialPunctuation`, `ucFinalPunctuation`, `ucOtherPunctuation`). The character can be specified as a UTF16 character `AChar` or a UTF16 encoded character starting at position `AIndex` in string `AString`.

Errors: If `AIndex` is not a valid character index in the string `AString`, an `EArgumentOutOfRangeException (256)` exception is raised. If the character at that position is not complete, an `EArgumentOutOfRangeException (256)` exception is raised.

See also: [IsControl \(266\)](#), [IsDigit \(267\)](#), [IsSurrogate \(267\)](#), [IsHighSurrogate \(268\)](#), [IsLowSurrogate \(268\)](#), [IsSurrogatePair \(269\)](#), [IsLetter \(269\)](#), [IsLetterOrDigit \(269\)](#), [IsLower \(270\)](#), [IsNumber \(270\)](#), [IsSymbol \(272\)](#), [IsUpper \(272\)](#), [IsWhiteSpace \(272\)](#)

2.5.19 TCharacter.IsSeparator

Synopsis: Check if a Unicode character is a separator character.

Declaration:

```
class function IsSeparator(AChar: UnicodeChar) : Boolean; Overload
                        ; Static
class function IsSeparator(const AString: UnicodeString;
                        AIndex: Integer) : Boolean; Overload; Static
```

Visibility: public

Description: `IsSeparator` returns `True` if a Unicode character has category that is one of the separator categories (`ucSpaceSeparator`, `ucLineSeparator`, `ucParagraphSeparator`). The character can be specified as a UTF16 character `AChar` or a UTF16 encoded character starting at position `AIndex` in string `AString`.

Errors: If `AIndex` is not a valid character index in the string `AString`, an `EArgumentOutOfRangeException (256)` exception is raised. If the character at that position is not complete, an `EArgumentOutOfRangeException (256)` exception is raised.

See also: [IsControl \(266\)](#), [IsDigit \(267\)](#), [IsSurrogate \(267\)](#), [IsHighSurrogate \(268\)](#), [IsLowSurrogate \(268\)](#), [IsSurrogatePair \(269\)](#), [IsLetter \(269\)](#), [IsLetterOrDigit \(269\)](#), [IsLower \(270\)](#), [IsNumber \(270\)](#), [IsPunctuation \(271\)](#), [IsSymbol \(272\)](#), [IsUpper \(272\)](#), [IsWhiteSpace \(272\)](#)

2.5.20 TCharacter.IsSymbol

Synopsis: Check if a Unicode character is a symbol character.

```
Declaration: class function IsSymbol(AChar: UnicodeChar) : Boolean; Overload
              ; Static
              class function IsSymbol(const AString: UnicodeString; AIndex: Integer)
              : Boolean; Overload; Static
```

Visibility: public

Description: `IsSymbol` returns `True` if a Unicode character has category that is one of the symbol categories (`ucMathSymbol`, `ucCurrencySymbol`, `ucModifierSymbol`, `ucOtherSymbol`). The character can be specified as a UTF16 character `AChar` or a UTF16 encoded character starting at position `AIndex` in string `AString`.

Errors: If `AIndex` is not a valid character index in the string `AString`, an `EArgumentOutOfRangeException` (256) exception is raised. If the character at that position is not complete, an `EArgumentException` (256) exception is raised.

See also: [IsControl \(266\)](#), [IsDigit \(267\)](#), [IsSurrogate \(267\)](#), [IsHighSurrogate \(268\)](#), [IsLowSurrogate \(268\)](#), [IsSurrogatePair \(269\)](#), [IsLetter \(269\)](#), [IsLetterOrDigit \(269\)](#), [IsLower \(270\)](#), [IsNumber \(270\)](#), [IsPunctuation \(271\)](#), [IsSeparator \(271\)](#), [IsUpper \(272\)](#), [IsWhiteSpace \(272\)](#)

2.5.21 TCharacter.IsUpper

Synopsis: Check whether a Unicode character is an uppercase letter.

[illegible]

Visibility: public

Description: IsUpper returns True if a Unicode character has category `ucUppercaseLetter`. The character can be specified as a UTF16 character `AChar` or a UTF16 encoded character starting at position `AIndex` in string `AString`.

Errors: If `AIndex` is not a valid character index in the string `AString`, an `EArgumentOutOfRangeException` (256) exception is raised. If the character at that position is not complete, an `EArgumentException` (256) exception is raised.

See also: [IsControl \(266\)](#), [IsDigit \(267\)](#), [IsSurrogate \(267\)](#), [IsHighSurrogate \(268\)](#), [IsLowSurrogate \(268\)](#), [IsSurrogatePair \(269\)](#), [IsLetter \(269\)](#), [IsLetterOrDigit \(269\)](#), [IsLower \(270\)](#), [IsNumber \(270\)](#), [IsPunctuation \(271\)](#), [IsSeparator \(271\)](#), [IsSymbol \(272\)](#), [IsWhiteSpace \(272\)](#)

2.5.22 TCharacter.IsWhiteSpace

Synopsis: Check whether a Unicode character is a whitespace character.

```
Declaration: class function IsWhiteSpace(AChar: UnicodeChar) : Boolean; Overload
                ; Static
    class function IsWhiteSpace(const AString: UnicodeString;
                                AIndex: Integer) : Boolean; Overload
                ; Static
```

Visibility: public

Description: `IsUpper` returns `True` if a Unicode character has is a whitespace character. It checks the character properties. The character can be specified as a UTF16 character `AChar` or a UTF16 encoded character starting at position `AIndex` in string `AString`.

Errors: If `AIndex` is not a valid character index in the string `AString`, an `EArgumentOutOfRangeException` (256) exception is raised. If the character at that position is not complete, an `EArgumentOutOfRangeException` (256) exception is raised.

See also: `IsControl` (266), `IsDigit` (267), `IsSurrogate` (267), `IsHighSurrogate` (268), `IsLowSurrogate` (268), `IsSurrogatePair` (269), `IsLetter` (269), `IsLetterOrDigit` (269), `IsLower` (270), `IsNumber` (270), `IsPunctuation` (271), `IsSeparator` (271), `IsSymbol` (272), `IsUpper` (272)

2.5.23 TCharacter.ToLower

Synopsis: Convert a character or string to lowercase.

Declaration:

```
class function ToLower(AChar: UnicodeChar) : UnicodeChar; Overload
; Static
class function ToLower(const AString: UnicodeString) : UnicodeString
; Overload; Static
class function ToLower(const AString: UnicodeString;
const AOptions: TCharacterOptions) : UnicodeString
; Overload; Static
```

Visibility: public

Description: `ToLower` converts the Unicode character `AChar` or string `AString` to lowercase. Options determines the behaviour of the conversion: if `AOptions` contains `coIgnoreInvalidSequence` then no exception will be raised when the string or character contains an invalid Unicode sequence. The default behaviour is to raise an `EArgumentOutOfRangeException` (256) exception when this happens.

Errors: If an invalid character is encountered, an `EArgumentOutOfRangeException` (256) exception is raised, unless `coIgnoreInvalidSequence` is specified in the options.

See also: `TCharacter.ToUpper` (273), `TCharacter.IsLower` (270), `TCharacter.IsUpper` (272)

2.5.24 TCharacter.ToUpper

Synopsis: Convert a character or string to uppercase.

Declaration:

```
class function ToUpper(AChar: UnicodeChar) : UnicodeChar; Overload
; Static
class function ToUpper(const AString: UnicodeString) : UnicodeString
; Overload; Static
class function ToUpper(const AString: UnicodeString;
const AOptions: TCharacterOptions) : UnicodeString
; Overload; Static
```

Visibility: public

Description: `ToUpper` converts the Unicode character `AChar` or string `AString` to uppercase. Options determines the behaviour of the conversion: if `AOptions` contains `coIgnoreInvalidSequence` then no exception will be raised when the string or character contains an invalid Unicode sequence. The default behaviour is to raise an `EArgumentOutOfRangeException` (256) exception when this happens.

Errors: If an invalid character is encountered, an `EArgumentOutOfRangeException` (256) exception is raised, unless `coIgnoreInvalidSequence` is specified in the options.

See also: `TCharacter.ToUpper` ([273](#)), `TCharacter.IsLower` ([270](#)), `TCharacter.IsUpper` ([272](#))

Chapter 3

Reference for unit 'charset'

3.1 Used units

Table 3.1: Used units by unit 'charset'

Name	Page
System	1335

3.2 Overview

The charset unit can be used to load single-byte character set (code page) descriptions. It is used in the `fpwistring` ([275](#)) unit to add support for converting single-byte codepage strings to Unicode strings (and vice versa).

Data of a code page may be included using one of the ready-made units, or can be loaded (in a binary form) at runtime with the `loadbinaryunicodemapping` ([277](#)) function. The binary files have the `.bcm` extension and are produced by the `creumap` utility distributed with Free Pascal.

Pre-made units are available for the following codepages: `cp895` ([275](#)), `cp932` ([275](#)), `cp936` ([275](#)), `cp949` ([275](#)) and `cp950` ([275](#)).

3.3 Constants, types and variables

3.3.1 Constants

```
BINARY_MAPPING_FILE_EXT = '.bcm'
```

`BINARY_MAPPING_FILE_EXT` contains the default extension of a file containing a binary-coded map.

3.3.2 Types

```
preversecharmapping = ^treversecharmapping
```

Pointer to `treversecharmapping`.

```
punicodecharmapping = ^tunicodecharmapping
```

Pointer to tunicodecharmapping.

```
punicodemap = ^tunicodemap
```

Pointer to tunicodemap.

```
tunicodechar = Word
```

tunicodechar is a type used to represent Unicode characters in this file, it should not be used for other Unicode routines.

```
tunicodecharmappingflag = (umf_noinfo, umf_leadbyte, umf_undefined,
    umf_unused)
```

Table 3.2: Enumeration values for type tunicodecharmappingflag

Value	Explanation
umf_leadbyte	Unicode character uses leading byte.
umf_noinfo	No extra information about Unicode character.
umf_undefined	Currently unused.
umf_unused	Unused position in code page.

tunicodecharmappingflag contains various Flags describing information about a Unicode character.

```
tunicodestring = ^tunicodechar
```

tunicodestring is a type used to represent Unicode strings in this file, it should not be used for other Unicode routines.

3.4 Procedures and functions

3.4.1 getascii

Synopsis: Convert Unicode character or string to single-byte character or string.

Declaration:

```
function getascii(c: tunicodechar; p: punicodemap) : string
function getascii(c: tunicodechar; p: punicodemap; ABuffer: PAnsiChar;
    ABufferLen: LongInt) : LongInt
```

Visibility: default

Description: getascii converts a Unicode character *c* to one or more single-byte characters according to the map in *p*. The result can be a string containing up to 2 characters, or the number of characters copied to the buffer *ABuffer* with length *ABufferLen*.

Errors: If the character cannot be translated, ASCII character 63 is returned (or copied to the buffer). In the case of the buffer variant of the function, -1 is then returned. If the buffer is not large enough, -1 is returned.

See also: [getunicode \(277\)](#)

3.4.2 getmap

Synopsis: Find a codepage map.

Declaration: `function getmap(const s: string) : puniconemap`
`function getmap(cp: Word) : puniconemap`

Visibility: default

Description: `getmap` looks in the registered codepage mappings and returns the mapping for the requested codepage. The codepage can be specified using a name `s` or a numerical identifier `cp`. The search is case sensitive.

Errors: if the requested map is not found, `Nil` is returned.

See also: `registermapping` (279), `registerbinarymapping` (278), `mappingavailable` (278)

3.4.3 getunicode

Synopsis: Map single-byte character to Unicode character.

Declaration: `function getunicode(c: Char; p: puniconemap) : tuniconedchar`
`function getunicode(AAnsiStr: PAnsiChar; AAnsiLen: LongInt;`
`AMap: puniconemap; ADest: tuniconestring) : LongInt`

Visibility: default

Description: The first form of `getunicode` will map a single character `c` to its Unicode equivalent for mapping `p`. If no equivalent can be found, 0 is returned.

The second form of `getunicode` will transform a string (specified using a pointer `AAnsiStr` to a buffer with length `AAnsiLen`) to a Unicode string using single byte codepage map `AMap`. It returns the number of Unicode characters. If `ADest` is `Nil` then just the number of characters is returned. If `ADest` is not `nil`, it must point to a buffer large enough to contain the Unicode string, and the converted string will be copied to it.

Errors: No checking on the validity of the buffers is done.

See also: `getascii` (276)

3.4.4 loadbinaryunicodemapping

Synopsis: Load binary single-byte codepage to Unicode map from file or memory.

Declaration: `function loadbinaryunicodemapping(const directory: string;`
`const cpname: string) : puniconemap`
`; Overload`
`function loadbinaryunicodemapping(const filename: string) : puniconemap`
`; Overload`
`function loadbinaryunicodemapping(const AData: Pointer;`
`const ADataLength: Integer)`
`: puniconemap; Overload`

Visibility: default

Description: `loadbinaryunicodemapping` loads a binary description of a single-byte Unicode mapping. The mapping can reside in a file, in which case the file to load can be specified using a filename `filename` or using a directory `directory` and codepage name `cpname`. In the latter case, a

suffix consisting of `_le` or `_be` depending on the endianness of the current platform will be appended, and the filename extension is `.bcm`. Note that the file names may be case sensitive.

The data can also be loaded from a memory block `AData` with size `ADataLength`.

It will produce an in-memory map of the file. It returns a pointer to the map, or `Nil` if something went wrong. The resulting mapping can be registered using [registermapping \(279\)](#).

Errors: On error, `Nil` is returned.

See also: [loadunicodemapping \(278\)](#), [registermapping \(279\)](#)

3.4.5 loadunicodemapping

Synopsis: Load textual single-byte codepage to Unicode map from file.

Declaration:

```
function loadunicodemapping(const cpname: string; const f: string;
                           cp: Word) : puniconemap
```

Visibility: default

Description: `loadunicodemapping` loads a text description of a single-byte Unicode mapping. It will analyse the textual description in file `f`, and produce an in-memory map of the file. It returns a pointer to the map, or `Nil` if something went wrong. The Unicode map name must be specified in `cpName`, and the numerical identifier in `cp`.

The resulting mapping can be registered using [registermapping \(279\)](#).

Errors: On error, `Nil` is returned.

See also: [loadbinaryunicodemapping \(277\)](#), [registermapping \(279\)](#)

3.4.6 mappingavailable

Synopsis: Check if a mapping is available for a specified code page.

Declaration:

```
function mappingavailable(const s: string) : Boolean
function mappingavailable(cp: Word) : Boolean
```

Visibility: default

Description: `mappingavailable` returns `True` if a mapping for a specified code page (using name `s` or numerical identifier `cp`) is available, or `False` if it is not.

See also: [registermapping \(279\)](#), [registerbinarymapping \(278\)](#), [getmap \(277\)](#)

3.4.7 registerbinarymapping

Synopsis: Load and register binary single-byte codepage to Unicode map from file.

Declaration:

```
function registerbinarymapping(const directory: string;
                              const cpname: string) : Boolean
```

Visibility: default

Description: `registerbinarymapping` calls [loadbinaryunicodemapping \(277\)](#) using `directory` and `cpname` and registers the resulting mapping, if any was successfully loaded, using [registermapping \(279\)](#). It returns `True` if the operation was successful.

Errors: On error, `False` is returned.

See also: [loadbinaryunicodemapping \(277\)](#), [registermapping \(279\)](#)

3.4.8 registermapping

Synopsis: Register mapping.

Declaration: `procedure registermapping(p: punicomemap)`

Visibility: default

Description: `RegisterMapping` registers mapping `p` in the registry of single-byte codepages. No attempt is made to avoid double registrations. In case of doubles, the last registered mapping will be used first.

See also: `loadunicodemapping` (278), `loadbinaryunicodemapping` (277), `registerbinarymapping` (278)

3.5 treversecharmapping

```
treversecharmapping = packed record
  unicode : tunicodechar;
  char1
    : Byte;
  char2 : Byte;
end
```

`treversecharmapping` describes how a Unicode character can be created in terms of single-byte characters.

3.6 TSerializedMapHeader

```
TSerializedMapHeader = packed record
  cpName : string;
  cp : UInt16
  ;
  mapLength : UInt32;
  lastChar : Int32;
  reverseMapLength : UInt32
  ;
end
```

`TSerializedMapHeader` is a record describing the binary map data file. The contents of this record can be found at offset zero of a (.bcm) file containing a single-byte Unicode map.

3.7 tunicodecharmapping

```
tunicodecharmapping = packed record
  unicode : tunicodechar;
  flag
    : tunicodecharmappingflag;
  reserved : Byte;
end
```

`tunicodecharmapping` describes a Unicode character. An array of these mappings is built for each character in the single-byte character set,

3.8 tunicodemap

```
tunicodemap = record
  cpname : string;
  cp : Word;
  map : punicodecharmapping
;
  lastchar : LongInt;
  reversemap : preversecharmapping;
  reversemaplength
  : LongInt;
  next : tunicodemap;
  internalmap : Boolean;
end
```

`tunicodemap` describes a complete mapping between a single-byte code page and a Unicode character set. It contains both a forward and backward mapping.

Chapter 4

Reference for unit 'Classes'

4.1 Used units

Table 4.1: Used units by unit 'Classes'

Name	Page
rtlconsts	??
System	1335
sysutils	1609
Types	2007
TypeInfo	2059

4.2 Overview

This documentation describes the FPC `classes` unit. The `Classes` unit contains basic classes for the Free Component Library (FCL):

- a `TList` ([426](#)) class for maintaining lists of pointers,
- `TStringList` ([481](#)) for lists of strings,
- `TCollection` ([376](#)) to manage collections of objects
- `TStream` ([466](#)) classes to support streaming.

Furthermore it introduces methods for object persistence, and classes that understand an owner-owned relationship, with automatic memory management.

4.3 Constants, types and variables

4.3.1 Constants

`dupAccept` = `Types.dupAccept`

Duplicate values can be added to the list.

```
dupError = Types.dupError
```

If an attempt is made to add a duplicate value to the list, an `EStringListError` (322) exception is raised.

```
dupIgnore = Types.dupIgnore
```

Duplicate values will not be added to the list, but no error will be triggered.

```
FilerSignature : Array[1..4] of Char = 'TPF0'
```

Constant that is found at the start of a binary stream containing a streamed component.

```
FilerSignature1 : Array[1..4] of Char = 'TPF1'
```

```
fmCreate = $FF00
```

`TFileStream.Create` (406) creates a new file if needed.

```
fmOpenRead = 0
```

`TFileStream.Create` (406) opens a file with read-only access.

```
fmOpenReadWrite = 2
```

`TFileStream.Create` (406) opens a file with read-write access.

```
fmOpenWrite = 1
```

`TFileStream.Create` (406) opens a file with write-only access.

```
MaxBitFlags = High(SizeInt) - bitsizeof(TBitsBase) - 1
```

Maximum number of bits in `TBits` collection.

```
MaxListSize = Maxint div 16
```

This constant sets the maximum number of elements in a `TList` (426).

```
scAlt = $8000
```

Indicates ALT key in a keyboard shortcut.

```
scCommand = $1000
```

```
scCtrl = $4000
```

indicates CTRL key in a keyboard shortcut.

`scNone = 0`

Indicates no special key is pressed in a keyboard shortcut.

`scShift = $2000`

Indicates Shift key in a keyboard shortcut.

`SGUIDObserved = '{663C603C-3F3C-4CC5-823C-AC8079F979E5}'`

Observed interface GUID as a string.

`SGUIDObserver = '{BC7376EA-199C-4C2A-8684-F4805F0691CA}'`

Observer interface GUID as a string.

`soFromBeginning = 0`

Seek (468) starts relative to the stream origin.

`soFromCurrent = 1`

Seek (468) starts relative to the current position in the stream.

`soFromEnd = 2`

Seek (468) starts relative to the stream end.

`ssCommand = ssMeta`

`toEOF = Char(0)`

Value returned by `TParser.Token` (446) when the end of the input stream was reached.

`toFloat = Char(4)`

Value returned by `TParser.Token` (446) when a floating point value was found in the input stream.

`toInteger = Char(3)`

Value returned by `TParser.Token` (446) when an integer was found in the input stream.

`toString = Char(2)`

Value returned by `TParser.Token` (446) when a string was found in the input stream.

`toSymbol = Char(1)`

Value returned by `TParser.Token` (446) when a symbol was found in the input stream.

`toWString = Char(5)`

Value returned by `TParser.Token` (446) when a widestring was found in the input stream.

4.3.2 Types

`EListError = SysUtils.EListError`

If an error occurs in one of the `TList` (426) or `TStrings` (486) methods, then a `EListError` exception is raised. This can occur in one of the following cases:

1. There is not enough memory to expand the list.
2. The list tried to grow beyond its maximal capacity.
3. An attempt was made to reduce the capacity of the list below the current element count.
4. An attempt was made to set the list count to a negative value.
5. A non-existent element of the list was referenced. (i.e. the list index was out of bounds)
6. An attempt was made to move an item to a position outside the list's bounds.

`HMODULE = PtrInt`

FPC doesn't support modules yet, so this is a dummy type.

`HRSRC = TFPResourceHandle deprecated`

This type is provided for Delphi compatibility, it is used for resource streams.

`PBitsBase = ^TBitsBase`

`PPointerList = ^TPointerList`

Pointer to an array of pointers.

`PStringItem = ^TStringItem`

Pointer to a `TStringItem` (318) record.

`PStringItemList = ^TStringItemList`

Pointer to a `TStringItemList` (297).

`TActionEvent = procedure (Action: TBasicAction; var Handled: Boolean
)
of object`

`TActiveXRegType = (axrComponentOnly, axrIncludeDescendants)`

Table 4.2: Enumeration values for type `TActiveXRegType`

Value	Explanation
<code>axrComponentOnly</code>	
<code>axrIncludeDescendants</code>	

This type is provided for compatibility only, and is currently not used in Free Pascal.

```
TAlignment = (taLeftJustify, taRightJustify, taCenter)
```

Table 4.3: Enumeration values for type TAlignment

Value	Explanation
taCenter	Text is displayed centered.
taLeftJustify	Text is displayed aligned to the left.
taRightJustify	Text is displayed aligned to the right.

The TAlignment type is used to specify the alignment of the text in controls that display a text.

```
TAncestorNotFoundEvent = procedure (Reader: TReader;
  const ComponentName: string;
  ComponentClass: TPersistentClass;
  var Component: TComponent) of object
```

This event occurs when an ancestor component cannot be found.

```
TBasicActionClass = Class of TBasicAction
```

TBasicAction ([349](#)) class reference.

```
TBasicActionLinkClass = Class of TBasicActionLink
```

TBasicActionLink ([353](#)) class reference.

```
TBiDiMode = (bdLeftToRight, bdRightToLeft, bdRightToLeftNoAlign,
  bdRightToLeftReadingOnly)
```

Table 4.4: Enumeration values for type TBiDiMode

Value	Explanation
bdLeftToRight	Texts read from left to right.
bdRightToLeft	Texts read from right to left.
bdRightToLeftNoAlign	Texts read from right to left, but not right-aligned.
bdRightToLeftReadingOnly	Texts read from right to left.

TBiDiMode describes bi-directional support for displaying texts.

```
TBitsBase = PtrInt
```

```
TBitsBaseUnsigned = PtrUInt
```

```
TCollectionItemClass = Class of TCollectionItem
```

`TCollectionItemClass` is used by the `TCollection.ItemClass` (381) property of `TCollection` (376) to identify the descendant class of `TCollectionItem` (383) which should be created and managed.

```
TCollectionNotification = (cnAdded,cnExtracting,cnDeleting)
```

Table 4.5: Enumeration values for type `TCollectionNotification`

Value	Explanation
<code>cnAdded</code>	An item is added to the collection.
<code>cnDeleting</code>	An item is deleted from the collection.
<code>cnExtracting</code>	An item is extracted from the collection.

`TCollectionNotification` is used in the `TCollection` (376) class to send notifications about changes to the collection.

```
TCollectionSortCompare = function(Item1: TCollectionItem;
    Item2: TCollectionItem) : Integer
```

`TCollectionSortCompare` is the prototype for a callback used in the `TCollection.Sort` (381) method. The procedure should compare `Item1` and `Item2` and return an integer:

Result < 0 if `Item1` comes before `Item2`

Result = 0 if `Item1` is at the same level as `Item2`

Result > 0 if `Item1` comes after `Item2`

```
TComponentClass = Class of TComponent
```

The `TComponentClass` type is used when constructing `TComponent` (386) descendant instances and when registering components.

```
TComponentName = String
```

Names of components are of type `TComponentName`. By specifying a different type, the Object inspector can handle this property differently than a standard string property.

```
TComponentState= Set of (csLoading,csReading,csWriting,csDestroying
    ,
                                csDesigning,csAncestor,csUpdating,csFixups
    ,
                                csFreeNotification,csInline,csDesignInstance
    )
```

Table 4.6: Enumeration values for type

Value	Explanation
<code>csAncestor</code>	The component is being streamed as part of a frame (?).
<code>csDesigning</code>	The component is being designed in an IDE.
<code>csDesignInstance</code>	??
<code>csDestroying</code>	The component is being destroyed.
<code>csFixups</code>	The component's references to other components are being fixed.
<code>csFreeNotification</code>	Indicates whether the component has freenotifications.
<code>csInline</code>	Component is part of a frame (?).
<code>csLoading</code>	The component is being loaded from the stream.
<code>csReading</code>	Properties are being read from the stream.
<code>csUpdating</code>	The component is being updated. This value is provided for Delphi compatibility and not used in FPC.
<code>csWriting</code>	Properties are being written to the stream.

The following values are possible:

csLoading The component (and all child components) are being loaded from a stream. This means that a `TReader` (451) instance is reading properties from this and child components from a stream and is applying the values found in the stream to the properties.

csReading The properties of this component are being read from a stream. This means that a `TReader` (451) instance is reading properties from this component from a stream and is applying the values.

csWriting The properties of this component are being written to a stream. This means that a `TWriter` (537) instance is writing properties from this component to a stream.

csDestroying The component is being destroyed.

csDesigning The component is being designed in an IDE.

csAncestor The component has a design ancestor. This is used to record differences between a component and its design ancestor. For example a form `TForm2` inherited from a form `TForm1`. `TForm1` and all its components are copied to `TForm2`. `TForm2` and all its inherited components have `csAncestor` set. Only differences between `TForm1` and `TForm2` are stored in the stream of `TForm2`. The child components of a frame put onto a form have `csAncestor` too.

csInline The component is a nested top level component. For example a frame on a form. The children of the frame do not have `csInline`, unless they are other frames.

csDesignInstance The component is designed (`csDesigning`) and is a root component, meaning it has no owner (`Owner=nil`).

csFixups The component's references to other components are being fixed. While reading a component from stream, it can happen that the stream contains a component reference property with a name of a component that was not yet created and read from the stream. Such properties are saved, and the missing references are resolved when the complete stream was read. This resolving step is called fixing up references, and the `csFixups` flag is set during this step.

csFreeNotification This flag indicates that the component has free notifications registered with `TComponent.FreeNotification` (390)


```
TComponentStyle= Set of (csInheritable,csCheckPropAvail,csSubComponent
,
                        csTransient)
```

Table 4.7: Enumeration values for type

Value	Explanation
csCheckPropAvail	??
csInheritable	The component can be on inherited forms.
csSubComponent	Subcomponent - streamed as part of the owning component.
csTransient	Transient component.

Describes the style of the component.

```
TCreateComponentEvent = procedure(Reader: TReader;
  ComponentClass: TComponentClass;
  var Component: TComponent) of object
```

Event handler type, occurs when a component instance must be created when a component is read from a stream.

```
TDataModuleClass = Class of TDataModule
```

TDataModuleClass defines the class pointer for TDataModule (400).

```
TDuplicates = Types.TDuplicates
```

Type to describe what to do with duplicate values in a TStringlist (481).

```
TExceptionClass = Class of Exception
```

TExceptionClass is the class pointer for the Exception (1821) class, defined in the SysUtils (1609) unit.

```
TFilerFlag = (ffInherited,ffChildPos,ffInline)
```

Table 4.8: Enumeration values for type TFilerFlag

Value	Explanation
ffChildPos	The position of the child on it's parent is included.
ffInherited	Stored object is an inherited object.
ffInline	Used for frames.

The TFiler class uses this enumeration type to decide whether the streamed object was streamed as part of an inherited form or not.

```
TFilerFlags = Set of TFilerFlag
```

Set of `TFilerFlag` (288).

```
TFindAncestorEvent = procedure(Writer: TWriter; Component: TComponent
;
                                const Name: string;
var Ancestor: TComponent;
var RootAncestor: TComponent) of object
```

Event that occurs w.

```
TFindComponentClassEvent = procedure(Reader: TReader;
const ClassName: string;
var ComponentClass: TComponentClass)
of object
```

Event handler type, occurs when a component class pointer must be found when reading a component from a stream.

```
TFindGlobalComponent = function(const Name: string) : TComponent
```

`TFindGlobalComponent` is a callback used to find a component in a global scope. It is used when the streaming system needs to find a component which is not part of the component which is currently being streamed. It should return the component with name `Name`, or `Nil` if none is found.

The variable `FindGlobalComponent` (306) is a callback of type `TFindGlobalComponent`. It can be set by the IDE when an unknown reference is found, to offer the designer to redirect the link to a new component.

```
TFindMethodEvent = procedure(Reader: TReader; const MethodName: string
;
                                var Address: CodePointer;
var Error: Boolean) of object
```

If a `TReader` (451) instance needs to locate a method and it doesn't find it in the streamed form, then the `OnFindMethod` (462) event handler will be called, if one is installed. This event can be assigned in order to use different locating methods. If a method is found, then its address should be returned in `Address`. The `Error` should be set to `True` if the reader should raise an exception after the event was handled. If it is set to `False` no exception will be raised, even if no method was found. On entry, `Error` will be set to `True`.

```
TFPObservedOperation = (ooChange, ooFree, ooAddItem, ooDeleteItem, ooCustom
)
```

Table 4.9: Enumeration values for type `TFPObservedOperation`

Value	Explanation
<code>ooAddItem</code>	An item is added to the observed object (generally a list).
<code>ooChange</code>	The observed object has changed.
<code>ooCustom</code>	Custom event.
<code>ooDeleteItem</code>	An item is deleted from the observed object (generally a list).
<code>ooFree</code>	The observed object is being freed.

`TFPObservedOperation` enumerates the possible operations that can be reported to an observer. Which of these operations is reported depends on the implementation of the observed object.

`TGetChildProc = procedure (Child: TComponent) of object`

Callback used when obtaining child components.

`TGetStrProc = procedure (const S: string) of object`

This event is used as a callback to retrieve string values. It is used, among other things, to pass along string properties in property editors.

`THandle = System.THandle`

This type is used as the handle for `THandleStream` (417) stream descendants

`THelpContext = - MaxLongint..MaxLongint`

Range type to specify help contexts.

`THelpEvent = function (Command: Word; Data: LongInt;
var CallHelp: Boolean) : Boolean of object`

This event is used for display of online help.

`THelpType = (htKeyword, htContext)`

Table 4.10: Enumeration values for type `THelpType`

Value	Explanation
<code>htContext</code>	Help type: Context ID help.
<code>htKeyword</code>	Help type: Keyword help.

Enumeration type specifying the kind of help requested.

`THintEvent = procedure (var HintStr: string; var CanShow: Boolean)
of object`

`TIdentToInt = function (const Ident: string; var Int: LongInt) :
Boolean`

`TIdentToInt` is a callback used to look up identifiers (`Ident`) and return an integer value corresponding to this identifier (`Int`). The callback should return `True` if a value corresponding to integer `Ident` was found, `False` if not.

A callback of type `TIdentToInt` should be specified when an integer is registered using the `RegisterIntegerConsts` (315) call.

`TInitComponentHandler = function (Instance: TComponent;
RootAncestor: TClass) : Boolean`

`TInitComponentHandler` is a callback type. It is used in the `InitInheritedComponent` (309) call to initialize a component. Callbacks of this type are registered with the `RegisterInitComponentHandler` (315) call.

```
TIntToIdent = function(Int: LongInt; var Ident: string) : Boolean
```

`TIdentToInt` is a callback used to look up integers (`Ident`) and return an identifier (`Ident`) that can be used to represent this integer value in an IDE. The callback should return `True` if a value corresponding to integer `Ident` was found, `False` if not.

A callback of type `TIntToIdent` should be specified when an integer is registered using the `RegisterIntegerConsts` (315) call.

```
TLeftRight = taLeftJustify..taRightJustify
```

`TLeftRight` is a subrange type based on the `TAlignment` (285) enumerated type. It contains only the left and right alignment constants.

```
TListAssignOp = (laCopy, laAnd, laOr, laXor, laSrcUnique, laDestUnique
)
```

Table 4.11: Enumeration values for type `TListAssignOp`

Value	Explanation
<code>laAnd</code>	Remove all elements not first second list.
<code>laCopy</code>	Clear list and copy all strings from second list.
<code>laDestUnique</code>	Keep all elements that exists only in list2.
<code>laOr</code>	Add all elements from second (and optional third) list, eliminate duplicates.
<code>laSrcUnique</code>	Just keep all elements that exist only in source list.
<code>laXor</code>	Remove elements in second lists, Add all elements from second list not in first list.

This type determines what operation `TList.Assign` (432) or `TFPList.assign` (413) performs.

```
TListCallback = Types.TListCallback
```

`TListCallback` is the method callback prototype for the function that is passed to the `TFPList.ForEachCall` (414) call. The `data` argument will be filled with all the pointers in the list (one per call) and the `arg` argument is the `Arg` argument passed to the `ForEachCall` call.

```
TListNotification = (lnAdded, lnExtracted, lnDeleted)
```

Table 4.12: Enumeration values for type `TListNotification`

Value	Explanation
<code>lnAdded</code>	List change notification: Element added to the list.
<code>lnDeleted</code>	List change notification: Element deleted from the list.
<code>lnExtracted</code>	List change notification: Element extracted from the list.

Kind of list notification event.

```
TListSortCompare = function(Item1: Pointer; Item2: Pointer) :
    Integer
```

Callback type for the list sort algorithm.

```
TListStaticCallback = Types.TListStaticCallback
```

`TListCallback` is the procedural callback prototype for the function that is passed to the `TFPList.ForEachCall` (414) call. The `data` argument will be filled with all the pointers in the list (one per call) and the `arg` argument is the `Arg` argument passed to the `ForEachCall` call.

```
TMissingNameValueSeparatorAction = (mnvaValue, mnvaName, mnvaEmpty,
    mnvaError)
```

Table 4.13: Enumeration values for type `TMissingNameValueSeparatorAction`

Value	Explanation
<code>mnvaEmpty</code>	Consider it an empty line.
<code>mnvaError</code>	Raise an exception if the Name=Value functionality is used.
<code>mnvaName</code>	Consider the line a name with empty value.
<code>mnvaValue</code>	Consider the line a value.

`TMissingNameValueSeparatorAction` is the type of the `TStrings.MissingNameValueSeparatorAction` (507) property and determines what to do if in a `TStrings` (486) descendent the `NameValueSeparator` (508) is missing. It can have the following values:

`mnvaValue` Consider the line a value.

`mnvaName` Consider the line a name with empty value.

`mnvaEmpty` Consider it an empty line.

`mnvaError` Raise an exception if the Name=Value functionality is used.

```
TMissingNameValueSeparatorActions = Set of
    TMissingNameValueSeparatorAction
```

`TMissingNameValueSeparatorActions` is simply a set of `TMissingNameValueSeparatorAction` (292) values.

```
TNotifyCallBack = procedure(Sender: TObject; AData: Pointer)
```

`TNotifyCallBack` is used to notify about thread termination when using static callbacks. When called, the `Sender` will contain the thread which is terminating, and `AData` is the `AData` parameter passed to `TThread.ExecuteInThread` (524).

```
TNotifyEvent = procedure(Sender: TObject) of object
```

Most event handlers are implemented as a property of type `TNotifyEvent`. When this is set to a certain method of a class, when the event occurs, the method will be called, and the class that generated the event will pass itself along as the `Sender` argument.

TObjectTextEncoding = (oteDFM, oteLFM)

Table 4.14: Enumeration values for type TObjectTextEncoding

Value	Explanation
oteDFM	Characters are in DFM (Delphi) format: widechar encoded.
oteLFM	Characters are in LFM format: UTF-8 encoded.

TObjectTextEncoding is an enumerated type which denotes the encoding of non ASCII characters in an object stream file. It is needed for correct encoding when reading string values in the text stream.

TOperation = (opInsert, opRemove)

Table 4.15: Enumeration values for type TOperation

Value	Explanation
opInsert	A new component is being inserted in the child component list.
opRemove	A component is being removed from the child component list.

Operation of which a component is notified.

TPersistentClass = Class of TPersistent

TPersistentClass is the class reference type for the TPersistent (446) class.

TPoint = Types.TPoint

This record describes a coordinate. It is used to handle the Top (386) and Left (386) properties of TComponent (386).

X represents the X-Coordinate of the point described by the record. Y represents the Y-Coordinate of the point described by the record.

TPointerList = Array[0..MaxListSize-1] of Pointer

Type for an Array of pointers.

```

TPropertyNotFoundEvent = procedure(Reader: TReader;
  Instance: TPersistent;
  var PropName: string;
  IsPath: Boolean;
  var
    Handled: Boolean;
    var Skip: Boolean
  ) of object

```

`TPropertyNotFoundEvent` is the prototype for the `TReader.OnPropertyNotFound` (461) event. `Reader` is the sender of the event, `Instance` is the instance that is being streamed. `PropInfo` is a pointer to the RTTI information for the property being read. `Handled` should be set to `True` if the handler redirected the unknown property successfully, and `Skip` should be set to `True` if the value should be skipped. `IsPath` determines whether the property refers to a sub-property.

```
TReadComponentsProc = procedure(Component: TComponent) of object
```

Callback type when reading a component from a stream.

```
TReaderError = procedure(Reader: TReader; const Message: string;
    var Handled: Boolean) of object
```

Event handler type, called when an error occurs during the streaming.

```
TReaderProc = procedure(Reader: TReader) of object
```

The `TReaderProc` reader procedure is a callback procedure which will be used by a `TPersistent` (446) descendant to read user properties from a stream during the streaming process. The `Reader` argument is the writer object which can be used read properties from the stream.

```
TReadWriteStringPropertyEvent = procedure(Sender: TObject;
    const Instance: TPersistent;
    PropInfo: PPropInfo;
    var Content: string) of
    object
```

`TReadWriteStringPropertyEvent` is the prototype for the `TReader.OnReadStringProperty` (463) event handler. `Reader` is the sender of the event, `Instance` is the instance that is being streamed. `PropInfo` is a pointer to the RTTI information for the property being read. `Content` is the string as it was read from the stream.

```
TRect = Types.TRect
```

`TRect` describes a rectangle in space with its upper-left (in `(Top,Left>)`) and lower-right (in `(Bottom,Right)`) corners.

```
TReferenceNameEvent = procedure(Reader: TReader; var Name: string
    )
                                of object
```

Occurs when a named object needs to be looked up.

```
TSeekOrigin = (soBeginning, soCurrent, soEnd)
```

Table 4.16: Enumeration values for type `TSeekOrigin`

Value	Explanation
<code>soBeginning</code>	Offset is interpreted relative to the start of the stream.
<code>soCurrent</code>	Offset is interpreted relative to the current position in the stream.
<code>soEnd</code>	Offset is interpreted relative to the end of the stream.

Specifies the origin of the `TStream.Seek` (468) method.

```
TSetMethodPropertyEvent = procedure(Reader: TReader;
  Instance: TPersistent;
  PropInfo: PPropInfo;
  const TheMethodName: string;
  var Handled: Boolean) of object
```

TSetMethodPropertyEvent is the prototype for the TReader.OnSetMethodProperty ([462](#)) event. Reader is the sender of the event, Instance is the instance that is being streamed. PropInfo is a pointer to the RTTI information for the property being read, and TheMethodName is the name of the method that the property should be set to. Handled should be set to True if the handler set the property successfully.

```
TSetNameEvent = procedure(Reader: TReader; Component: TComponent;
  var Name: string) of object
```

Occurs when the reader needs to set a component's name.

```
TShiftState = Set of TShiftStateEnum
```

This type is used when describing a shortcut key or when describing what special keys are pressed on a keyboard when a key event is generated.

The set contains the special keys that can be used in combination with a 'normal' key.

```
TShiftStateEnum = (ssShift, ssAlt, ssCtrl, ssLeft, ssRight, ssMiddle,
  ssDouble, ssMeta, ssSuper, ssHyper, ssAltGr, ssCaps
  , ssNum,
  ssScroll, ssTriple, ssQuad, ssExtral, ssExtra2
  , ssTouch,
  ssPen, ssHorizontal)
```


Table 4.17: Enumeration values for type TShiftStateEnum

Value	Explanation
ssAlt	Alt key pressed.
ssAltGr	Alt-GR key pressed.
ssCaps	Caps lock key pressed.
ssCtrl	Ctrl key pressed.
ssDouble	Double mouse click.
ssExtra1	Extra key 1.
ssExtra2	Extra key 2.
ssHorizontal	
ssHyper	Hyper key pressed.
ssLeft	Left mouse button pressed.
ssMeta	Meta key pressed.
ssMiddle	Middle mouse button pressed.
ssNum	Num lock key pressed.
ssPen	
ssQuad	Quadruple mouse click.
ssRight	Right mouse button pressed.
ssScroll	Scroll lock key pressed.
ssShift	Shift key pressed.
ssSuper	Super key pressed.
ssTouch	
ssTriple	Triple mouse click.

Keyboard/Mouse shift state enumerator.

```
TShortCut = (Word) .. (Word)
```

Enumeration type to identify shortcut key combinations.

```
TSmallPoint = Types.TSmallPoint
```

Same as TPoint (293), only the X and Y ranges are limited to 2-byte integers instead of 4-byte integers.

```
TStreamOwnership = (soReference, soOwned)
```

Table 4.18: Enumeration values for type TStreamOwnership

Value	Explanation
soOwned	Stream is owned: it will be freed when the adapter is freed.
soReference	Stream is referenced only, it is not freed by the adapter.

The ownership of a streamadapter determines what happens with the stream on which a TStreamAdapter (476) acts, when the adapter is freed.

```
TStreamProc = procedure(Stream: TStream) of object
```

Procedure type used in streaming.

```
TStringItemList = Array[0..MaxListSize] of TStringItem
```

This declaration is provided for Delphi compatibility, it is not used in Free Pascal.

```
TStringListSortCompare = function(List: TStringList; Index1: Integer
;
                                Index2: Integer) : Integer
```

Callback type used in stringlist compares.

```
TStringsClass = Class of TStrings
```

TStringsClass is the class of TStrings (486). It is provided for convenience.

```
TStringsFilterMethod = function(const s: string) : Boolean of
    object
```

TStringsFilterMethod is the prototype for the callback in TStrings.Filter (496). Return True if the string S must be included in the TStrings.Filter (496) result, False if it must be excluded.

```
TStringsForEachMethod = procedure(const CurrentValue: string) of
    object
```

TStringsForEachMethod is a callback signature for TStrings.ForEach (496). It gets passed the string value in CurrentValue.

```
TStringsForEachMethodEx = procedure(const CurrentValue: string;
    const index: Integer) of
    object
```

TStringsForEachMethodEx is a callback signature for TStrings.ForEach (496). It gets passed the string value in CurrentValue, string index in Index.

```
TStringsForEachMethodExObj = procedure(const CurrentValue: string
;
                                const index: Integer;
    Obj: TObject) of object
```

TStringsForEachMethodExObj is a callback signature for TStrings.ForEach (496). It gets passed the string value in CurrentValue, string index in Index and associated object in Obj.

```
TStringsMapMethod = function(const s: string) : string of object
```

TStringsMapMethod is the prototype for the callback in TStrings.Map (500). The method must return the mapped value for value S.

```
TStringsOption = (soStrictDelimiter, soWriteBOM, soTrailingLineBreak
,
                    soUseLocale, soPreserveBOM)
```

Table 4.19: Enumeration values for type TStringsOption

Value	Explanation
soPreserveBOM	If this option is enabled, TStrings.WriteBOM (512) is set in TStrings.LoadFromStream (500) or TStrings.LoadFromFile (500).
soStrictDelimiter	See TStrings.StrictDelimiter (509).
soTrailingLineBreak	See TStrings.TrailingLineBreak (509).
soUseLocale	See TStrings.UseLocale (511).
soWriteBOM	See TStrings.WriteBOM (512).

TStringsOption is the type used in the TStrings.Options (508) set property, and it lists various strings options. It has the following values:

soStrictDelimiter See TStrings.StrictDelimiter (509).

soWriteBOM See TStrings.WriteBOM (512).

soTrailingLineBreak See TStrings.TrailingLineBreak (509).

soUseLocale See TStrings.UseLocale (511).

soPreserveBOM If this option is enabled, TStrings.WriteBOM (512) is set in TStrings.LoadFromStream (500) or TStrings.LoadFromFile (500) according to BOM presence in the loaded file.

TStringsOptions = Set of TStringsOption

TStringsOptions is the set type for enumeration TStringsOption (298).

```
TStringsReduceMethod = function(const s1: string; const s2: string
)
: string of object
```

TStringsReduceMethod is the prototype for the callback in TStrings.Reduce (501). The method must return the reduced value for values S1 and S2. On every iteration S1 will contain the result of the previous iteration.

TStringsSortStyle = (sslNone, sslUser, sslAuto)

Table 4.20: Enumeration values for type TStringsSortStyle

Value	Explanation
sslAuto	The TStrings instance keeps the strings sorted.
sslNone	The strings are not sorted.
sslUser	The strings are kept sorted by the user.

TStringsSortStyle is the type used in the TStringList.SortStyle (486) property, and describes how the strings are sorted when the TStringList.Sorted (485) property is True. It has the following values:

sslNone The strings are not sorted.

sslUser The strings are kept sorted by the user.

sslAuto The TStrings instance keeps the strings sorted.

TStringsSortStyles = Set of TStringsSortStyle

TStringsSortStyles is the set type for enumeration TStringsSortStyle (298).

TSynchronizeProcVar = procedure

Synchronize callback type.

TThreadExecuteCallBack = procedure(AData: Pointer)

TThreadExecuteCallBack is the signature of the static procedure to be used when executing something in a thread using TThread.ExecuteInThread (524) when no status reporting is required.

TThreadExecuteHandler = TThreadMethod

TThreadExecuteHandler is the signature of the method to be used when executing something in a thread using TThread.ExecuteInThread (524) when no status reporting is required.

```
TThreadExecuteStatusCallBack = procedure(AData: Pointer;
    ReportStatus: TThreadReportStatus
)
```

TThreadExecuteStatusCallBack is the signature of a procedure to be used when executing something in a thread using TThread.ExecuteInThread (524) when status reporting is required.

On entry in the method, AData is the AData parameter passed to TThread.ExecuteInThread (524). ReportStatus is passed to the method, and the threaded procedure can call ReportStatus at various stages to report about the status of the method: The status will be reported to the main thread using TThread.synchronize (520), so calls to ReportStatus will be blocked as long as the status was not reported.

```
TThreadExecuteStatusHandler = procedure
    (ReportStatus: TThreadReportStatus)
    of object
```

TThreadExecuteStatusHandler is the signature of the method to be used when executing something in a thread using TThread.ExecuteInThread (524) when status reporting is required.

On entry in the method, ReportStatus is passed to the method, and the method can call ReportStatus at various stages to report about the status of the method: The status will be reported to the main thread using TThread.synchronize (520), and calls ReportStatus will be blocked as long as the status was not reported.

TThreadMethod = procedure of object

Procedure variable used when synchronizing threads.

```
TThreadPriority = (tpIdle, tpLowest, tpLower, tpNormal, tpHigher, tpHighest
    ,
    tpTimeCritical)
```

Table 4.21: Enumeration values for type TThreadPriority

Value	Explanation
tpHigher	Thread runs at high priority.
tpHighest	Thread runs at highest possible priority.
tpIdle	Thread only runs when other processes are idle.
tpLower	Thread runs at a lower priority.
tpLowest	Thread runs at the lowest priority.
tpNormal	Thread runs at normal process priority.
tpTimeCritical	Thread runs at realtime priority.

Enumeration specifying the priority at which a thread runs.

`TThreadReportStatus = procedure(const status: string) of object`

`TThreadReportStatus` is the callback prototype for the `TThread.ExecuteInThread` (524) method. This callback is used to report thread status to the main thread: the `Status` string can be used to report the status of thread execution.

`TThreadStatusNotifyCallBack = procedure(Sender: TThread;
AData: Pointer;
const status: string)`

`TThreadStatusNotifyCallBack` is the signature of the callback to be provided when executing a static procedure in a thread using `TThread.ExecuteInThread` (524) when status reporting is required.

This callback will be called in the main thread. When called, it has the thread whose status is reported in `Sender`, `AData` is the `AData` parameter passed to `TThread.ExecuteInThread` (524). Finally, the status message in `ReportStatus` is passed to the callback.

Note that the thread reporting its status is blocked while the callback is being handled: The status is be reported to the main thread using `TThread.synchronize` (520).

`TThreadStatusNotifyEvent = procedure(Sender: TThread;
const status: string) of object`

`TThreadStatusNotifyEvent` is the signature of the method to be provided when executing something in a thread using `TThread.ExecuteInThread` (524) when status reporting is required.

This event handler will be called in the main thread. When called, it has the thread whose status is reported in `Sender`, and the status message in `ReportStatus` is passed to the method.

Note that the thread reporting its status is blocked while the callback is being handled: The status is be reported to the main thread using `TThread.synchronize` (520).

`TTopBottom = taAlignTop..taAlignBottom`

`TTopBottom` is a subrange of `TVerticalAlignment` (301) that excludes the vertically centered alignment value.

`TValueType = (vaNull, vaList, vaInt8, vaInt16, vaInt32, vaExtended, vaString
,`

```

    vaIdent, vaFalse, vaTrue, vaBinary, vaSet, vaLString, vaNil
,
    vaCollection, vaSingle, vaCurrency, vaDate, vaWString
, vaInt64,
    vaUTF8String, vaUString, vaQWord, vaDouble)

```

Table 4.22: Enumeration values for type TValueType

Value	Explanation
vaBinary	Binary data follows.
vaCollection	Collection follows.
vaCurrency	Currency value follows.
vaDate	Date value follows.
vaDouble	
vaExtended	Extended value.
vaFalse	Boolean <code>False</code> value.
vaIdent	Identifier.
vaInt16	Integer value, 16 bits long.
vaInt32	Integer value, 32 bits long.
vaInt64	Integer value, 64 bits long.
vaInt8	Integer value, 8 bits long.
vaList	Identifies the start of a list of values.
vaLString	Ansistring data follows.
vaNil	Nil pointer.
vaNull	Empty value. Ends a list.
vaQWord	QWord (64-bit word) value.
vaSet	Set data follows.
vaSingle	Single type follows.
vaString	String value.
vaTrue	Boolean <code>True</code> value.
vaUString	UnicodeString value.
vaUTF8String	UTF8 encoded Unicode string.
vaWString	Widestring value follows.

Enumerated type used to identify the kind of streamed property.

```
TVerticalAlignment = (taAlignTop, taAlignBottom, taVerticalCenter)
```

Table 4.23: Enumeration values for type TVerticalAlignment

Value	Explanation
taAlignBottom	Align vertically to the bottom.
taAlignTop	Align vertically to the top.
taVerticalCenter	Align vertically centered.

`TVerticalAlignment` can be used to specify a vertical alignment. It is not used in the classes `unit`, and is provided for symmetry with `TAlignment` (285). It can have the following values:

taAlignTop Align vertically to the top.

taAlignBottom Align vertically to the bottom.

taVerticalCenter Align vertically centered.

```
TWriteMethodPropertyEvent = procedure(Writer: TWriter;
  Instance: TPersistent;
  PropInfo: PPropInfo;
  const MethodValue: TMethod;
  const DefMethodValue: TMethod;
  var Handled: Boolean) of object
```

TWriteMethodPropertyEvent is the prototype for the TWriter.OnWriteMethodProperty (544) event. Writer is the sender of the event, Instance is the instance that is being streamed. PropInfo is a pointer to the RTTI information for the property being written, and MethodValue is the value of the method that the property was set to. DefMethodCodeValue is set to the default value of the property (Nil or the parent value). Handled should be set to True if the handler set the property successfully.

```
TWriterProc = procedure(Writer: TWriter) of object
```

The TWriterProc writer procedure is a callback procedure which will be used by a TPersistent (446) descendant to write user properties from a stream during the streaming process. The Writer argument is the writer object which can be used write properties to the stream.

4.3.3 Variables

```
AddDataModule : procedure(DataModule: TDataModule) of object
```

AddDataModule can be set by an IDE or a streaming mechanism to receive notification when a new instance of a TDataModule (400) descendant is created.

```
ApplicationHandleException : procedure(Sender: TObject) of object
```

ApplicationHandleException can be set by an application object to handle any exceptions that may occur when a TDataModule (400) is created.

```
ApplicationShowException : procedure(E: Exception) of object
```

Unused.

```
CreateVCLComObjectProc : procedure(Component: TComponent) = Nil
```

CreateVCLComObjectProc is called by TComponent if it needs to create a IVCLComObject interface for itself (when the ComObject property is read). It passes itself as the Component parameter.

```
GlobalNameSpace : IReadWriteSync
```

An interface protecting the global namespace. Used when reading/writing to the global namespace list during streaming of forms.

```
MainThreadID : TThreadID
```

ID of main thread. Unused at this point.

```
RegisterComponentsProc : procedure(const Page: string;
    ComponentClasses: Array of TComponentClass)
```

RegisterComponentsProc can be set by an IDE to be notified when new components are being registered. Application programmers should never have to set RegisterComponentsProc

```
RegisterNoIconProc : procedure(ComponentClasses: Array of TComponentClass
    )
```

RegisterNoIconProc can be set by an IDE to be notified when new components are being registered, and which do not need an Icon in the component palette. Application programmers should never have to set RegisterComponentsProc

```
RemoveDataModule : procedure(DataModule: TDataModule) of object
```

RemoveDataModule can be set by an IDE or a streaming mechanism to receive notification when an instance of a TDataModule (400) descendant is freed.

```
WakeMainThread : TNotifyEvent = Nil
```

WakeMainThread is a handler, which, when set, is called by the TThread.Synchronize (520) routine to signal the main thread that a synchronization routine is waiting in the queue.

This handler is by default empty. An actual implementation depends on the main program logic (usually an event loop) and must be provided by the event loop logic: the event loop will normally call CheckSynchronize (304) at regular intervals. The WakeMainThread can make sure this happens as soon as possible.

While this handle should alert the main program thread that a thread is waiting for synchronization, the call is executed by the thread, and should therefore NOT synchronize the thread, but should somehow signal the main thread that a thread is waiting for synchronization. For example, by sending a message.

4.4 Procedures and functions

4.4.1 ActivateClassGroup

Synopsis: Activates a class group.

Declaration: `function ActivateClassGroup(AClass: TPersistentClass) : TPersistentClass`

Visibility: default

Description: ActivateClassGroup activates the group of classes to which AClass belongs. The function returns the class that was last used to activate the class group.

The class registration and streaming mechanism allows to organize the classes in groups. This allows an IDE to form groups of classes, which can be enabled or disabled. It is not needed at Run-Time.

Errors: If AClass does not belong to a class group, an exception is raised.

See also: StartClassGroup (317), GroupDescendentsWith (308), ClassGroupOf (305)

4.4.2 BeginGlobalLoading

Synopsis: Not yet implemented.

Declaration: `procedure BeginGlobalLoading`

Visibility: default

Description: Not yet implemented.

4.4.3 BinToHex

Synopsis: Convert a binary buffer to a hexadecimal string.

Declaration: `procedure BinToHex (BinValue: PChar; HexValue: PChar;
BinBufSize: Integer)`

Visibility: default

Description: `BinToHex` converts the byte values in `BinValue` to a string consisting of 2-character hexadecimal strings in `HexValue`. `BufSize` specifies the length of `BinValue`, which means that `HexValue` must have size $2 * \text{BufSize}$.

For example a buffer containing the byte values 255 and 0 will be converted to FF00.

Errors: No length checking is done, so if an invalid size is specified, an exception may follow.

See also: `HexToBin` ([308](#))

4.4.4 Bounds

Synopsis: Returns a `TRect` structure with the bounding rect of the given location and size.

Declaration: `function Bounds (ALeft: Integer; ATop: Integer; AWidth: Integer;
AHeight: Integer) : TRect`

Visibility: default

Description: `Bounds` returns a `TRect` ([294](#)) record with the given origin (`ALeft`, `ATop`) and dimensions (`AWidth`, `AHeight`) filled in. The bottom-right corner is calculated by adding `AWidth` to `ALeft` and `AHeight` to `ATop`. As a result, a rectangle with width/height set to 0 is exactly 1 pixel.

See also: `Rect` ([313](#))

4.4.5 CheckSynchronize

Synopsis: Check whether there are any synchronize calls in the synchronize queue.

Declaration: `function CheckSynchronize (timeout: LongInt) : Boolean`

Visibility: default

Description: `CheckSynchronize` should be called regularly by the main application thread to handle any `TThread.Synchronize` ([520](#)) calls that may be waiting for execution by the main thread. If any such calls are waiting for execution by the main thread, they are executed at once, in the order that they were scheduled.

The function returns `True` if any `Synchronize` method was executed.

`TimeOut` is the maximum amount of time (in milliseconds) that the `CheckSynchronize` routine will wait for synchronisation requests to appear in the queue.

Calling this routine more often will ensure that synchronize requests are handled faster.

This routine may not be called from any thread other than the main thread, as it will execute the waiting requests.

Threads may call the `WakeMainThread` (303) to signal the main thread that the synchronisation queue contains items, and thus speed up the execution of the synchronize calls.

See also: `TThread.Synchronize` (520), `WakeMainThread` (303)

4.4.6 ClassGroupOf

Synopsis: Returns the class group to which an instance or class belongs.

Declaration: `function ClassGroupOf(AClass: TPersistentClass) : TPersistentClass`
`function ClassGroupOf(Instance: TPersistent) : TPersistentClass`

Visibility: default

Description: `ClassGroupOf` returns the class group to which `AClass` or `Instance` belongs.

Errors: The result is `Nil` if no matching class group is found.

See also: `StartClassGroup` (317), `ActivateClassGroup` (303), `GroupDescendentsWith` (308)

4.4.7 CollectionsEqual

Synopsis: Returns `True` if two collections are equal.

Declaration: `function CollectionsEqual(C1: TCollection; C2: TCollection) : Boolean`
`function CollectionsEqual(C1: TCollection; C2: TCollection;`
`Owner1: TComponent; Owner2: TComponent)`
`: Boolean`

Visibility: default

Description: `CollectionsEqual` is not yet implemented. It simply returns `False`

4.4.8 EndGlobalLoading

Synopsis: Not yet implemented.

Declaration: `procedure EndGlobalLoading`

Visibility: default

Description: Not yet implemented.

4.4.9 ExtractStrings

Synopsis: Split a string in different words.

Declaration: `function ExtractStrings(Separators: TSysCharSet;`
`WhiteSpace: TSysCharSet; Content: PChar;`
`Strings: TStrings; AddEmptyStrings: Boolean)`
`: Integer`

Visibility: default

Description: `ExtractStrings` splits `Content` (a null-terminated string) into words, and adds the words to the `Strings` stringlist. The words are separated by `Separators` and any characters in whitespace are stripped from the strings. The space and CR/LF characters are always considered whitespace.

Errors: No length checking is performed on `Content`. If no null-termination character is present, an access violation may occur. Likewise, if `Strings` is not valid, an access violation may occur.

4.4.10 FindClass

Synopsis: Returns the class pointer of a class with given name.

Declaration: `function FindClass(const AClassName: string) : TPersistentClass`

Visibility: default

Description: `FindClass` searches for the class named `ClassName` in the list of registered classes and returns a class pointer to the definition. If no class with the given name could be found, an exception is raised.

The `GetClass` (307) function does not raise an exception when it does not find the class, but returns a `Nil` pointer instead.

See also: `RegisterClass` (313), `GetClass` (307)

4.4.11 FindGlobalComponent

Synopsis: Callback used when a component must be found.

Declaration: `function FindGlobalComponent(const Name: string) : TComponent`

Visibility: default

Description: `FindGlobalComponent` is a callback of type `TFindGlobalComponent` (289). It can be set by the IDE when an unknown reference is found, to offer the user to redirect the link to a new component.

It is a callback used to find a component in a global scope. It is used when the streaming system needs to find a component which is not part of the component which is currently being streamed. It should return the component with name `Name`, or `Nil` if none is found.

See also: `TFindGlobalComponent` (289)

4.4.12 FindIdentToInt

Synopsis: Return the string to integer converter for an integer type.

Declaration: `function FindIdentToInt(AIntegerType: Pointer) : TIdentToInt`

Visibility: default

Description: `FindIdentToInt` returns the handler that handles the conversion of a string representation to an integer that can be used in component streaming, when `IdentToInt` (309) is called.

Errors: `Nil` is returned if no handler is registered for the given type.

4.4.13 FindIntToIdent

Synopsis: Return the integer to string converter for an integer type.

Declaration: `function FindIntToIdent (AIntegerType: Pointer) : TIntToIdent`

Visibility: default

Description: `FindIntToIdent` returns the handler that handles the conversion of an integer to a string representation that can be used in component streaming, when `IntToIdent` (310) is called.

Errors: Nil is returned if no handler is registered for the given type.

See also: `IntToIdent` (310), `TIntToIdent` (291), `FindIdentToInt` (306)

4.4.14 FindNestedComponent

Synopsis: Finds the component with name path starting at the indicated root component.

Declaration: `function FindNestedComponent (Root: TComponent; APath: string;
CStyle: Boolean) : TComponent`

Visibility: default

Description: `FindNestedComponent` will descend through the list of owned components (starting at `Root`) and will return the component whose name path matches `NamePath`. As a path separator the characters `.` (dot), `-` (dash) and `>` (greater than) can be used

See also: `GlobalFixupReferences` (308)

4.4.15 GetClass

Synopsis: Returns the class pointer of a class with given name.

Declaration: `function GetClass (const AClassName: string) : TPersistentClass`

Visibility: default

Description: `GetClass` searches for the class named `ClassName` in the list of registered classes and returns a class pointer to the definition. If no class with the given name could be found, Nil is returned.

The `FindClass` (306) function will raise an exception if it does not find the class.

See also: `RegisterClass` (313), `GetClass` (307)

4.4.16 GetFixupInstanceNames

Synopsis: Returns the names of elements that need to be resolved for the `root` component, whose reference contains `ReferenceRootName`.

Declaration: `procedure GetFixupInstanceNames (Root: TComponent;
const ReferenceRootName: string;
Names: TStrings)`

Visibility: default

Description: `GetFixupInstanceNames` examines the list of unresolved references and returns the names of classes that contain unresolved references to the `Root` component in the list `Names`. The list is not cleared prior to filling it.

See also: `GetFixupReferenceNames` (308), `GlobalFixupReferences` (308)

4.4.17 GetFixupReferenceNames

Synopsis: Returns the names of elements that need to be resolved for the `root` component.

Declaration: `procedure GetFixupReferenceNames (Root: TComponent; Names: TStrings)`

Visibility: `default`

Description: `GetFixupReferenceNames` examines the list of unresolved references and returns the names of properties that must be resolved for the component `Root` in the list `Names`. The list is not cleared prior to filling it.

See also: `GetFixupInstanceNames` (307), `GlobalFixupReferences` (308)

4.4.18 GlobalFixupReferences

Synopsis: Called to resolve unresolved references after forms are loaded.

Declaration: `procedure GlobalFixupReferences`

Visibility: `default`

Description: `GlobalFixupReferences` runs over the list of unresolved references and tries to resolve them. This routine should under normal circumstances not be called in an application programmer's code. It is called automatically by the streaming system after a component has been instantiated and its properties read from a stream. It will attempt to resolve references to other global components.

See also: `GetFixupReferenceNames` (308), `GetFixupInstanceNames` (307)

4.4.19 GroupDescendentsWith

Synopsis: Provided for Delphi compatibility.

Declaration: `procedure GroupDescendentsWith (AClass: TPersistentClass;
AClassGroup: TPersistentClass)`

Visibility: `default`

Description: `GroupDescendentsWith` exists for Delphi compatibility, it doesn't actually do anything in Free Pascal.

See also: `RegisterClasses` (314)

4.4.20 HexToBin

Synopsis: Convert a hexadecimal string to a binary buffer.

Declaration: `function HexToBin (HexValue: PChar; BinValue: PChar; BinBufSize: Integer)
: Integer`

Visibility: `default`

Description: `HexToBin` scans the hexadecimal string representation in `HexValue` and transforms every 2 character hexadecimal number to a byte and stores it in `BinValue`. The buffer size is the size of the binary buffer. Scanning will stop if the size of the binary buffer is reached or when an invalid character is encountered. The return value is the number of stored bytes.

Errors: No length checking is done, so if an invalid size is specified, an exception may follow.

See also: `BinToHex` (304)

4.4.21 IdentToInt

Synopsis: Looks up an integer value in a integer-to-identifier map list.

Declaration: `function IdentToInt(const Ident: string; out Int: LongInt;
const Map: Array of TIdentMapEntry) : Boolean`

Visibility: default

Description: `IdentToInt` searches `Map` for an entry whose `Name` field matches `Ident` and returns the corresponding integer value in `Int`. If a match was found, the function returns `True`, otherwise, `False` is returned.

See also: `TIdentToInt` (290), `TIntToIdent` (291), `IntToIdent` (310), `TIdentMapEntry` (318)

4.4.22 IfThen

Declaration: `function IfThen(AValue: Boolean; const ATrue: TStringList;
const AFalse: TStringList) : TStringList; Overload`

Visibility: default

4.4.23 InitComponentRes

Synopsis: Provided for Delphi compatibility only.

Declaration: `function InitComponentRes(const ResName: string; Instance: TComponent)
: Boolean`

Visibility: default

Description: This function is provided for Delphi compatibility. It always returns `false`.

See also: `ReadComponentRes` (312)

4.4.24 InitInheritedComponent

Synopsis: Initializes a component descending from `RootAncestor`.

Declaration: `function InitInheritedComponent(Instance: TComponent;
RootAncestor: TClass) : Boolean`

Visibility: default

Description: `InitInheritedComponent` should be called from a constructor to read properties of the component `Instance` from the streaming system. The `RootAncestor` class is the root class from which `Instance` is a descendant. This must be one of `TDatamodule`, `TCustomForm` or `TFrame`. The function returns `True` if the properties were successfully read from a stream or `False` if some error occurred.

See also: `ReadComponentRes` (312), `ReadComponentResEx` (312), `ReadComponentResFile` (313)

4.4.25 IntToIdent

Synopsis: Looks up an identifier for an integer value in a identifier-to-integer map list.

Declaration: `function IntToIdent (Int: LongInt; var Ident: string;
const Map: Array of TIdentMapEntry) : Boolean`

Visibility: default

Description: `IntToIdent` searches `Map` for an entry whose `Value` field matches `Int` and returns the corresponding identifier in `Ident`. If a match was found, the function returns `True`, otherwise, `False` is returned.

See also: `TIdentToInt` (290), `TIntToIdent` (291), `IdentToInt` (309), `TIdentMapEntry` (318)

4.4.26 InvalidPoint

Synopsis: Check whether a point is invalid.

Declaration: `function InvalidPoint (X: Integer; Y: Integer) : Boolean
function InvalidPoint (const At: TPoint) : Boolean
function InvalidPoint (const At: TSmallPoint) : Boolean`

Visibility: default

Description: `InvalidPoint` returns `True` if the X and Y coordinates (of the `TPoint` or `TSmallPoint` records, if one of these versions is used) are -1.

See also: `TPoint` (293), `TSmallPoint` (296), `PointsEqual` (312)

4.4.27 LineStart

Synopsis: Finds the start of a line in `Buffer` before `BufPos`.

Declaration: `function LineStart (Buffer: PChar; BufPos: PChar) : PChar`

Visibility: default

Description: `LineStart` reversely scans `Buffer` starting at `BufPos` for a linefeed character. It returns a pointer at the linefeed character.

4.4.28 NotifyGlobalLoading

Synopsis: Not yet implemented.

Declaration: `procedure NotifyGlobalLoading`

Visibility: default

Description: Not yet implemented.

4.4.29 ObjectBinaryToText

Synopsis: Converts an object stream from a binary to a text format.

```

Declaration: procedure ObjectBinaryToText (Input: TStream; Output: TStream;
                                             Encoding: TObjectTextEncoding)
               procedure ObjectBinaryToText (Input: TStream; Output: TStream)

```

Visibility: default

Description: `ObjectBinaryToText` reads an object stream in binary format from `Input` and writes the object stream in text format to `Output`. No components are instantiated during the process, this is a pure conversion routine.

See also: [ObjectTextToBinary \(311\)](#)

4.4.30 ObjectResourceToText

Synopsis: Converts an object stream from a (windows) resource to a text format.

Declaration: procedure ObjectResourceToText (Input: TStream; Output: TStream)

Visibility: default

Description: `ObjectResourceToText` reads the resource header from the `Input` stream and then passes the streams to `ObjectBinaryToText` (311)

See also: [ObjectBinaryToText \(311\)](#), [ObjectTextToResource \(311\)](#)

4.4.31 ObjectTextToBinary

Synopsis: Converts an object stream from a text to a binary format.

Declaration: `procedure ObjectTextToBinary(Input: TStream; Output: TStream)`

Visibility: default

Description: Converts an object stream from a text to a binary format.

4.4.32 ObjectTextToResource

Synopsis: Converts an object stream from a text to a (windows) resource format.

Declaration: `procedure ObjectTextToResource (Input: TStream; Output: TStream)`

Visibility: default

Description: `ObjectTextToResource` reads an object stream in text format from `Input` and writes a resource stream to `Output`.

Note that for the current implementation of this method in Free Pascal, the output stream should support positioning. (e.g. it should not be a pipe)

See also: [ObjectBinaryToText \(311\)](#), [ObjectResourceToText \(311\)](#)

4.4.33 Point

Synopsis: Returns a `TPoint` record with the given coordinates.

Declaration: `function Point (AX: Integer; AY: Integer) : TPoint`

Visibility: default

Description: `Point` returns a `TPoint` (293) record with the given coordinates `AX` and `AY` filled in.

See also: `TPoint` (293), `SmallPoint` (316), `Rect` (313), `Bounds` (304)

4.4.34 PointsEqual

Synopsis: Check whether two `TPoint` variables are equal.

Declaration: `function PointsEqual (const P1: TPoint; const P2: TPoint) : Boolean`
`function PointsEqual (const P1: TSmallPoint; const P2: TSmallPoint)`
`: Boolean`

Visibility: default

Description: `PointsEqual` compares the `P1` and `P2` points (of type `TPoint` (293) or `TSmallPoint` (296)) and returns `True` if the `X` and `Y` coordinates of the points are equal, or `False` otherwise.

See also: `TPoint` (293), `TSmallPoint` (296), `InvalidPoint` (310)

4.4.35 ReadComponentRes

Synopsis: Read component properties from a resource in the current module.

Declaration: `function ReadComponentRes (const ResName: string; Instance: TComponent)`
`: TComponent`

Visibility: default

Description: `ReadComponentRes` will read the component's properties from the resource `ResName` in the current module (always program module). It returns `Nil` if the resource was not found. It returns `Instance` if the resource was found and successfully applied to the component.

Errors: The function may raise an exception if the stream contains wrong data.

See also: `ReadComponentResEx` (312)

4.4.36 ReadComponentResEx

Synopsis: Read component properties from a resource in the specified module.

Declaration: `function ReadComponentResEx (HInstance: THandle; const ResName: string)`
`: TComponent`

Visibility: default

Description: `ReadComponentRes` will locate the resource `ResName` in instance `HInstance` (the current program, normally). It returns `Nil` if the resource was not found. It returns an instantiated component with all properties found in the stream, applied. This requires that the component is registered using `registerclass`.

Errors: The function may raise an exception if the stream contains wrong data.

See also: `ReadComponentRes` (312)

4.4.37 ReadComponentResFile

Synopsis: Read component properties from a specified resource file.

Declaration: `function ReadComponentResFile(const FileName: string;
Instance: TComponent) : TComponent`

Visibility: default

Description: `ReadComponentResFile` starts reading properties for `Instance` from the file `FileName`. It creates a filestream from `FileName` and then calls the `TStream.ReadComponentRes` (470) method to read the state of the component from the stream.

See also: `TStream.ReadComponentRes` (470), `WriteComponentResFile` (318)

4.4.38 Rect

Synopsis: Returns a `TRect` record with the given coordinates.

Declaration: `function Rect(ALeft: Integer; ATop: Integer; ARight: Integer;
ABottom: Integer) : TRect`

Visibility: default

Description: `Rect` returns a `TRect` (294) record with the given top-left (`ALeft`, `ATop`) and bottom-right (`ABottom`, `ARight`) corners filled in.

No checking is done to see whether the coordinates are valid.

See also: `TRect` (294), `Point` (312), `SmallPoint` (316), `Bounds` (304)

4.4.39 RedirectFixupReferences

Synopsis: Redirects references under the `root` object from `OldRootName` to `NewRootName`.

Declaration: `procedure RedirectFixupReferences(Root: TComponent;
const OldRootName: string;
const NewRootName: string)`

Visibility: default

Description: `RedirectFixupReferences` examines the list of unresolved references and replaces references to a root object named `OldRootName` with references to root object `NewRootName`.

An application programmer should never need to call `RedirectFixupReferences`. This function can be used by an IDE to support redirection of broken component links.

See also: `RemoveFixupReferences` (316)

4.4.40 RegisterClass

Synopsis: Registers a class with the streaming system.

Declaration: `procedure RegisterClass(AClass: TPersistentClass)`

Visibility: default

Description: `RegisterClass` registers the class `AClass` in the streaming system. After the class has been registered, it can be read from a stream when a reference to this class is encountered.

See also: `RegisterClasses` (314), `RegisterClassAlias` (314), `RegisterComponents` (314), `UnregisterClass` (317)

4.4.41 RegisterClassAlias

Synopsis: Registers a class alias with the streaming system.

Declaration: `procedure RegisterClassAlias (AClass: TPersistentClass;
const Alias: string)`

Visibility: default

Description: `RegisterClassAlias` registers a class alias in the streaming system. If a reference to a class `Alias` is encountered in a stream, then an instance of the class `AClass` will be created instead by the streaming code.

See also: `RegisterClass` (313), `RegisterClasses` (314), `RegisterComponents` (314), `UnregisterClass` (317)

4.4.42 RegisterClasses

Synopsis: Registers multiple classes with the streaming system.

Declaration: `procedure RegisterClasses (AClasses: Array of TPersistentClass)`

Visibility: default

Description: `RegisterClasses` registers the specified classes `AClass` in the streaming system. After the classes have been registered, they can be read from a stream when a reference to this class is encountered.

See also: `RegisterClass` (313), `RegisterClassAlias` (314), `RegisterComponents` (314), `UnregisterClass` (317)

4.4.43 RegisterComponents

Synopsis: Registers components for the component palette.

Declaration: `procedure RegisterComponents (const Page: string;
ComponentClasses: Array of TComponentClass)`

Visibility: default

Description: `RegisterComponents` registers the component on the appropriate component page. The component pages can be used by an IDE to display the known components so an application programmer may pick and use the components in his programs.

`Registercomponents` inserts the component class in the correct component page. If the `RegisterComponentsProc` procedure is set, this is called as well. Note that this behaviour is different from Delphi's behaviour where an exception will be raised if the procedural variable is not set.

See also: `RegisterClass` (313), `RegisterNoIcon` (315)

4.4.44 RegisterFindGlobalComponentProc

Synopsis: Register a component searching handler.

Declaration: `procedure RegisterFindGlobalComponentProc
(AFindGlobalComponent: TFindGlobalComponent`

Visibility: default

Description: `RegisterFindGlobalComponentProc` registers a global component search callback `AFindGlobalComponent`. When `FindGlobalComponent` (306) is called, then this callback will be used to search for the component.

Errors: None.

See also: `FindGlobalComponent` (306), `UnRegisterFindGlobalComponentProc` (317)

4.4.45 RegisterInitComponentHandler

Synopsis: Register a component initialization handler.

Declaration: `procedure RegisterInitComponentHandler(ComponentClass: TComponentClass;
Handler: TInitComponentHandler)`

Visibility: default

Description: `RegisterInitComponentHandler` registers a component initialization handler `Handler` for the component `ComponentClass`. This handler will be used to initialize descendants of `ComponentClass` in the `InitInheritedComponent` (309) call.

See also: `InitInheritedComponent` (309), `TInitComponentHandler` (291)

4.4.46 RegisterIntegerConsts

Synopsis: Registers some integer-to-identifier mappings.

Declaration: `procedure RegisterIntegerConsts(IntegerType: Pointer;
IdentToIntFn: TIdentToInt;
IntToIdentFn: TIntToIdent)`

Visibility: default

Description: `RegisterIntegerConsts` registers a pair of callbacks to be used when an integer of type `IntegerType` must be mapped to an identifier (using `IntToIdentFn`) or when an identifier must be mapped to an integer (using `IdentToIntFn`).

Component programmers can use `RegisterIntegerConsts` to associate a series of identifier strings with integer values for a property. A necessary condition is that the property should have a separate type declared using the `type integer` syntax. If a type of integer is defined in this way, an IDE can show symbolic names for the values of these properties.

The `IntegerType` should be a pointer to the type information of the integer type. The `IntToIdentFn` and `IdentToIntFn` are two callbacks that will be used when converting between the identifier and integer value and vice versa. The functions `IdentToInt` (309) and `IntToIdent` (310) can be used to implement these callback functions.

See also: `TIdentToInt` (290), `TIntToIdent` (291), `IdentToInt` (309), `IntToIdent` (310)

4.4.47 RegisterNoIcon

Synopsis: Registers components that have no icon on the component palette.

Declaration: `procedure RegisterNoIcon(ComponentClasses: Array of TComponentClass)`

Visibility: default

Description: `RegisterNoIcon` performs the same function as `RegisterComponents` (314) except that it calls `RegisterNoIconProc` (303) instead of `RegisterComponentsProc` (303)

See also: `RegisterNoIconProc` (303), `RegisterComponents` (314)

4.4.48 RegisterNonActiveX

Synopsis: Register non-activex component.

Declaration: `procedure RegisterNonActiveX
(ComponentClasses: Array of TComponentClass;
AxRegType: TActiveXRegType)`

Visibility: default

Description: Not yet implemented in Free Pascal

4.4.49 RemoveFixupReferences

Synopsis: Removes references to rootname from the fixup list.

Declaration: `procedure RemoveFixupReferences (Root: TComponent;
const RootName: string)`

Visibility: default

Description: `RemoveFixupReferences` examines the list of unresolved references and removes references to a root object pointing at `Root` or a root component named `RootName`.

An application programmer should never need to call `RemoveFixupReferences`. This function can be used by an IDE to support removal of broken component links.

See also: `RedirectFixupReferences` ([313](#))

4.4.50 RemoveFixups

Synopsis: Removes `Instance` from the fixup list.

Declaration: `procedure RemoveFixups (Instance: TPersistent)`

Visibility: default

Description: `RemoveFixups` removes all entries for component `Instance` from the list of unresolved references.

See also: `RedirectFixupReferences` ([313](#)), `RemoveFixupReferences` ([316](#))

4.4.51 SmallPoint

Synopsis: Returns a `TSmallPoint` record with the given coordinates.

Declaration: `function SmallPoint (AX: SmallInt; AY: SmallInt) : TSmallPoint`

Visibility: default

Description: `SmallPoint` returns a `TSmallPoint` ([296](#)) record with the given coordinates `AX` and `AY` filled in.

See also: `TSmallPoint` ([296](#)), `Point` ([312](#)), `Rect` ([313](#)), `Bounds` ([304](#))

4.4.52 StartClassGroup

Synopsis: Start new class group.

Declaration: `procedure StartClassGroup (AClass: TPersistentClass)`

Visibility: default

Description: `StartClassGroup` starts a new class group and adds `AClass` to it.

The class registration and streaming mechanism allows to organize the classes in groups. This allows an IDE to form groups of classes, which can be enabled or disabled. It is not needed at Run-Time.

See also: `GroupDescendentsWith` (308), `ActivateClassGroup` (303), `ClassGroupOf` (305)

4.4.53 UnRegisterClass

Synopsis: Unregisters a class from the streaming system.

Declaration: `procedure UnRegisterClass (AClass: TPersistentClass)`

Visibility: default

Description: `UnregisterClass` removes the class `AClass` from the class definitions in the streaming system.

See also: `UnRegisterClasses` (317), `UnRegisterModuleClasses` (318), `RegisterClass` (313)

4.4.54 UnRegisterClasses

Synopsis: Unregisters multiple classes from the streaming system.

Declaration: `procedure UnRegisterClasses (AClasses: Array of TPersistentClass)`

Visibility: default

Description: `UnregisterClasses` removes the classes in `AClasses` from the class definitions in the streaming system.

4.4.55 UnregisterFindGlobalComponentProc

Synopsis: Remove a previously registered component searching handler.

Declaration: `procedure UnregisterFindGlobalComponentProc`
`(AFindGlobalComponent: TFindGlobalComponent)`

Visibility: default

Description: `UnregisterFindGlobalComponentProc` unregisters the previously registered global component search callback `AFindGlobalComponent`. After this call, when `FindGlobalComponent` (306) is called, then this callback will be no longer be used to search for the component.

Errors: None.

See also: `FindGlobalComponent` (306), `RegisterFindGlobalComponentProc` (314)

4.4.56 UnRegisterModuleClasses

Synopsis: Unregisters classes registered by module.

Declaration: `procedure UnRegisterModuleClasses (Module: HMODULE)`

Visibility: default

Description: `UnRegisterModuleClasses` unregisters all classes which reside in the module `Module`. For each registered class, the definition pointer is checked to see whether it resides in the module, and if it does, the definition is removed.

See also: `UnRegisterClass` (317), `UnRegisterClasses` (317), `RegisterClasses` (314)

4.4.57 WriteComponentResFile

Synopsis: Write component properties to a specified resource file.

Declaration: `procedure WriteComponentResFile (const FileName: string;
Instance: TComponent)`

Visibility: default

Description: `WriteComponentResFile` starts writing properties of `Instance` to the file `FileName`. It creates a filestream from `FileName` and then calls `TStream.WriteComponentRes` (471) method to write the state of the component to the stream.

See also: `TStream.WriteComponentRes` (471), `ReadComponentResFile` (313)

4.5 TIdentMapEntry

```
TIdentMapEntry = record
  Value : Integer;
  Name  : string;
end
```

`TIdentMapEntry` is used internally by the `IdentToInt` (309) and `IntToIdent` (310) calls to store the mapping between the identifiers and the integers they represent.

4.6 TStringItem

```
TStringItem = record
  FString : string;
  FObject : TObject;
end
```

The `TStringItem` is used to store the string and object items in a `TStringList` (481) string list instance. It should never be used directly.

4.7 EBitsError

4.7.1 Description

When an index of a bit in a TBits ([368](#)) is out of the valid range (0 to Count-1) then a EBitsError exception is raised.

4.8 EClassNotFound

4.8.1 Description

When the streaming system needs to create a component, it looks for the class pointer (VMT) in the list of registered classes by its name. If this name is not found, then an EClassNotFound is raised.

See also: EFileError ([319](#))

4.9 EComponentError

4.9.1 Description

When an error occurs during the registration of a component, or when naming a component, then a EComponentError is raised. Possible causes are:

1. An name with an illegal character was assigned to a component.
2. A component with the same name and owner already exists.
3. The component registration system isn't set up properly.

See also: TComponent ([386](#)), TComponent.Name ([396](#))

4.10 EFCREATEError

4.10.1 Description

When the operating system reports an error during creation of a new file in the Filestream Constructor ([406](#)), a EFCREATEError is raised.

See also: EStreamError ([321](#)), EFOpenError ([320](#))

4.11 EFileError

4.11.1 Description

This class serves as an ancestor class for exceptions that are raised when an error occurs during component streaming. A EFileError exception is raised when a class is registered twice.

See also: EStreamError ([321](#)), EReadError ([321](#))

4.12 EOpenError

4.12.1 Description

When the operating system reports an error during the opening of a file in the Filestream Constructor (406), a `EOpenError` is raised.

See also: `EStreamError` (321), `ECreateError` (319)

4.13 EInvalidImage

4.13.1 Description

This exception is not used by Free Pascal but is provided for Delphi compatibility.

4.14 EInvalidOperation

4.14.1 Description

This exception is not used in Free Pascal, it is defined for Delphi compatibility purposes only.

4.15 EMethodNotFound

4.15.1 Description

This exception is no longer used in the streaming system. This error is replaced by a `EReadError` (321).

See also: `EFileError` (319), `EReadError` (321)

4.16 EObserver

4.16.1 Description

`EObserver` is an error that is raised when an object is registered as an observer, and it does not implement the `IFPObserver` (325) interface.

See also: `IFPObserver` (325), `IFPObserver.FPOAttachObserver` (324)

4.17 EOutOfResources

4.17.1 Description

This exception is not used in Free Pascal, it is defined for Delphi compatibility purposes only.

4.18 EParserError

4.18.1 Description

When an error occurs during the parsing of a stream, an `EParserError` is raised. Usually this indicates that an invalid token was found on the input stream, or the token read from the stream wasn't the expected token.

See also: `TParser` ([440](#))

4.19 EReadError

4.19.1 Description

If an error occurs when reading from a stream, a `EReadError` exception is raised. Possible causes for this are:

1. Not enough data is available when reading from a stream
2. The stream containing a component's data contains invalid data. this will occur only when reading a component from a stream.

See also: `EFileError` ([319](#)), `EWriteError` ([322](#))

4.20 EResNotFound

4.20.1 Description

This exception is not used by Free Pascal but is provided for Delphi compatibility.

4.21 EStreamError

4.21.1 Description

An `EStreamError` is raised when an error occurs during reading from or writing to a stream: Possible causes are

1. Not enough data is available in the stream.
2. Trying to seek beyond the beginning or end of the stream.
3. Trying to set the capacity of a memory stream and no memory is available.
4. Trying to write to a read-only stream, such as a resource stream.
5. Trying to read from a write-only stream.

See also: `EFCREATEError` ([319](#))

4.22 EStringListError

4.22.1 Description

When an error occurs in one of the methods of `TStrings` (486) then an `EStringListError` is raised. This can have one of the following causes:

1. There is not enough memory to expand the list.
2. The list tried to grow beyond its maximal capacity.
3. A non-existent element of the list was referenced. (i.e. the list index was out of bounds)
4. An attempt was made to add a duplicate entry to a `TStringList` (481) when `TStringList.Duplicates` (484) is `False`.

See also: `TStrings` (486), `TStringList` (481)

4.23 EThread

4.23.1 Description

Thread error exception.

4.24 EThreadDestroyCalled

4.24.1 Description

Exception raised when a thread is destroyed illegally.

4.25 EThreadExternalException

4.25.1 Description

`EThreadExternalException` is raised by for example `TThread.CheckTerminated` (522) and `TThread.SetReturnValue` (522) when the thread was not created by the Free Pascal program, but by an external code base (for example a DLL, or the calling application in a DLL).

See also: `TThread.CheckTerminated` (522), `TThread.SetReturnValue` (522)

4.26 EWriteError

4.26.1 Description

If an error occurs when writing to a stream, a `EWriteError` exception is raised. Possible causes for this are:

1. The stream doesn't allow writing.
2. An error occurred when writing a property to a stream.

See also: `EFileError` (319), `EReadError` (321)

4.27 IDesignerNotify

4.27.1 Description

`IDesignerNotify` is an interface that can be used to communicate changes to a designer mechanism. It offers functionality for detecting changes, and notifications when the component is destroyed.

4.27.2 Method overview

Page	Method	Description
323	<code>Modified</code>	Notify that the component is modified.
323	<code>Notification</code>	Notification of owner changes.

4.27.3 IDesignerNotify.Modified

Synopsis: Notify that the component is modified.

Declaration: `procedure Modified`

Visibility: `default`

Description: `Modified` can be used to notify a designer of changes, indicating that components should be streamed.

4.27.4 IDesignerNotify.Notification

Synopsis: Notification of owner changes.

Declaration: `procedure Notification(AnObject: TPersistent; Operation: TOperation)`

Visibility: `default`

Description: `Notification` is the interface counterpart of `TComponent.Notification` ([387](#)) which is used to communicate adds to the components.

See also: `TComponent.Notification` ([387](#))

4.28 IFPObserved

4.28.1 Description

`IFPObserved` is an interface which can be implemented in objects that must be observable. Objects that wish to observe the object can register themselves with the `FPOAttachObserver` ([281](#)) call, and must be detached using the `FPODetachObserver` ([281](#)) call.

This interface is not reference counted, so care must be taken that the `ooFree` message is sent with `FPONotifyObservers` ([281](#)) when the object is freed.

See also: `FPONotifyObservers` ([281](#))

4.28.2 Method overview

Page	Method	Description
324	FPOAttachObserver	Attach a new observer to the object.
324	FPODetachObserver	Remove an observer from the list of observers.
324	FPONotifyObservers	Notify all observers.

4.28.3 IFPObserved.FPOAttachObserver

Synopsis: Attach a new observer to the object.

Declaration: `procedure FPOAttachObserver(AObserver: TObject)`

Visibility: default

Description: `FPOAttachObserver` must be called with an object instance `AObserver` that implements the `IFPObserver` ([325](#)) interface. The `FPOObservedChanged` ([325](#)) method of the interface will be called whenever `FPONotifyObservers` ([281](#)) is used to notify observers of a change. Objects implementing this interface should check that `AObserver` actually implements the `IFPObserver` ([325](#)) interface.

Do not make assumptions on how the interface behaves if `FPOAttachObserver` is called more than once with the same interface. It may add the object to the list of observers unconditionally (in which case it will be notified twice) or it may check that it is not yet in the list.

Errors: If `AObserver` does not implement the `IFPObserver` ([325](#)) interface, an `EObserver` ([320](#)) exception must be raised. No other errors should be raised, other than a possible out of memory error.

See also: `IFPObserver` ([325](#)), `FPOObservedChanged` ([325](#)), `FPONotifyObservers` ([281](#))

4.28.4 IFPObserved.FPODetachObserver

Synopsis: Remove an observer from the list of observers.

Declaration: `procedure FPODetachObserver(AObserver: TObject)`

Visibility: default

Description: `FPODetachObserver` removes the `AObserver` object from the list of observers. If it was not in the list, then this is silently accepted. Once removed, it will no longer receive notifications when `FPOObservedChanged` ([325](#)) is called.

If the object was added more than once using `FPOAttachObserver` ([281](#)), then it depends on the implementor of the interface whether or `FPODetachObserver` must be called an equal number of times.

See also: `IFPObserver` ([325](#)), `FPOObservedChanged` ([325](#)), `FPONotifyObservers` ([281](#)), `FPOAttachObserver` ([281](#))

4.28.5 IFPObserved.FPONotifyObservers

Synopsis: Notify all observers.

Declaration: `procedure FPONotifyObservers(ASender: TObject;
 AOperation: TFPObservedOperation;
 Data: Pointer)`

Visibility: default

Description: `FPONotifyObservers` notifies all observers of the object that a change has occurred. It calls `FPOObservedChanged` (325) on the `IFPObserver` (325) interface of all attached objects, and passes on `ASender` (normally this is `Self`), `AOperation` and `Data`. What `Data` is, depends on the implementor of the interface.

There is no guaranteed order in which the change notifications are delivered to the observers. This is an implementation-specific detail, which should not be relied upon in any way.

See also: `IFPObserver` (325), `FPOObservedChanged` (325), `FPODetachObserver` (281), `FPOAttachObserver` (281)

4.29 IFPObserver

4.29.1 Description

`IFPObserver` is the interface an object must implement if it wishes to receive change notifications from another object. The presence of this interface will be checked when the object registers itself using `IFPObserver.FPOAttachObserver` (325). The change notifications arrive because the `FPOObservedChanged` (281) method is called by the observed object.

See also: `IFPObserver` (323), `FPOAttachObserver` (324)

4.29.2 Method overview

Page	Method	Description
325	<code>FPOObservedChanged</code>	Entry point for change notifications.

4.29.3 IFPObserver.FPOObservedChanged

Synopsis: Entry point for change notifications.

Declaration: `procedure FPOObservedChanged(ASender: TObject;
Operation: TFPObservedOperation;
Data: Pointer)`

Visibility: default

Description: `FPOObservedChanged` is the method that is called by an observed object (`IFPObserver` (323)) when it calls `FPONotifyObservers` (324). The `Sender` is the object under observation, the `Operation` and `Data` are the parameters used in the call to `FPONotifyObservers`.

See also: `IFPObserver` (323), `FPONotifyObservers` (324)

4.30 IInterfaceComponentReference

4.30.1 Description

`IInterfaceComponentReference` is an interface to return the component that implements a given interface. It is implemented by `TComponent` (386).

See also: `TComponent` (386)

4.30.2 Method overview

Page	Method	Description
326	<code>GetComponent</code>	Return component instance.

4.30.3 `IInterfaceComponentReference.GetComponent`

Synopsis: Return component instance.

Declaration: `function GetComponent : TComponent`

Visibility: default

Description: `GetComponent` returns the component instance.

Errors: None.

See also: `TComponent` ([386](#))

4.31 `IInterfaceList`

4.31.1 Description

`IInterfaceList` is an interface for maintaining a list of interfaces, strongly resembling the standard `TList` ([426](#)) class. It offers the same list of public methods as `TList`, with the exception that it uses interfaces instead of pointers.

All interfaces in the list should descend from `IUnknown`.

More detailed descriptions of how the various methods behave can be found in the `TList` reference.

See also: `TList` ([426](#))

4.31.2 Method overview

Page	Method	Description
329	<code>Add</code>	Add an interface to the list.
328	<code>Clear</code>	Clear the list.
328	<code>Delete</code>	Remove an interface from the list.
329	<code>Exchange</code>	Exchange 2 interfaces in the list.
329	<code>First</code>	Return the first non-empty interface in the list.
327	<code>Get</code>	Retrieve an interface pointer from the list.
327	<code>GetCapacity</code>	Return the capacity of the list.
327	<code>GetCount</code>	Return the current number of elements in the list.
329	<code>IndexOf</code>	Return the index of an interface.
329	<code>Insert</code>	Insert an interface in the list.
330	<code>Last</code>	Returns the last non-nil interface in the list.
330	<code>Lock</code>	Lock the list.
327	<code>Put</code>	Write an item to the list.
330	<code>Remove</code>	Remove an interface from the list.
328	<code>SetCapacity</code>	Set the capacity of the list.
328	<code>SetCount</code>	Set the number of items in the list.
330	<code>Unlock</code>	Unlock the list.

4.31.3 Property overview

Page	Properties	Access	Description
330	Capacity	rw	Capacity of the list.
331	Count	rw	Current number of elements in the list.
331	Items	rw	Provides Index-based, sequential, access to the interfaces in the list.

4.31.4 IInterfaceList.Get

Synopsis: Retrieve an interface pointer from the list.

Declaration: `function Get(i: Integer) : IUnknown`

Visibility: default

Description: `Get` returns the interface pointer at position `i` in the list. It serves as the `Read` method for the `Items` ([331](#)) property.

See also: `IInterfaceList.Items` ([331](#)), `TList.Items` ([434](#))

4.31.5 IInterfaceList.GetCapacity

Synopsis: Return the capacity of the list.

Declaration: `function GetCapacity : Integer`

Visibility: default

Description: `GetCapacity` returns the current capacity of the list. It serves as the `Read` method for the `Capacity` ([330](#)) property.

See also: `IInterfaceList.Capacity` ([330](#)), `TList.Capacity` ([433](#))

4.31.6 IInterfaceList.GetCount

Synopsis: Return the current number of elements in the list.

Declaration: `function GetCount : Integer`

Visibility: default

Description: It serves as the `Read` method for the `Count` ([331](#)) property.

See also: `IInterfaceList.Count` ([331](#)), `TList.Count` ([434](#))

4.31.7 IInterfaceList.Put

Synopsis: Write an item to the list.

Declaration: `procedure Put(i: Integer; item: IUnknown)`

Visibility: default

Description: `Put` writes the interface `Item` at position `I` in the list. It servers as the `Write` method for the `Items` ([331](#)) property.

See also: `IInterfaceList.Items` ([331](#)), `TList.Items` ([434](#))

4.31.8 IList.SetCapacity

Synopsis: Set the capacity of the list.

Declaration: `procedure SetCapacity(NewCapacity: Integer)`

Visibility: default

Description: `SetCapacity` sets the capacity of the list to `NewCapacity`. It serves as the `Write` method for the `Capacity` (330) property.

See also: `IInterfaceList.Capacity` (330), `TList.Capacity` (433)

4.31.9 IList.SetCount

Synopsis: Set the number of items in the list.

Declaration: `procedure SetCount(NewCount: Integer)`

Visibility: default

Description: `SetCount` sets the count of the list to `NewCount`. It serves as the `Write` method for the `Capacity` (330)

See also: `IInterfaceList.Count` (331), `TList.Count` (434)

4.31.10 IList.Clear

Synopsis: Clear the list.

Declaration: `procedure Clear`

Visibility: default

Description: `Clear` removes all interfaces from the list. All interfaces in the list will be cleared (i.e. their reference count will decrease with 1)

See also: `TList.Clear` (429)

4.31.11 IList.Delete

Synopsis: Remove an interface from the list.

Declaration: `procedure Delete(index: Integer)`

Visibility: default

Description: `Delete` removes the interface at position `Index` from the list. It does this by explicitly clearing the interface and then removing the slot.

See also: `TList.Clear` (429), `IInterfaceList.Add` (329), `IInterfaceList.Delete` (328), `IInterfaceList.Insert` (329)

4.31.12 IList.Exchange

Synopsis: Exchange 2 interfaces in the list.

Declaration: `procedure Exchange(index1: Integer; index2: Integer)`

Visibility: default

Description: `Exchange` exchanges 2 interfaces in the list at locations `index1` and `Index2`.

See also: `TList.Exchange` (430), `IInterfaceList.Add` (329), `IInterfaceList.Delete` (328), `IInterfaceList.Insert` (329)

4.31.13 IList.First

Synopsis: Return the first non-empty interface in the list.

Declaration: `function First : IUnknown`

Visibility: default

Description: `First` returns the first non-empty interface in the list.

See also: `TList.First` (431), `IInterfaceList.IndexOf` (329), `IInterfaceList.Last` (330)

4.31.14 IList.IndexOf

Synopsis: Return the index of an interface.

Declaration: `function IndexOf(const item: IUnknown) : Integer`

Visibility: default

Description: `IndexOf` returns the location in the list of the interface `Item`. If there is no such interface in the list, then -1 is returned.

See also: `TList.IndexOf` (431), `IInterfaceList.First` (329), `IInterfaceList.Last` (330)

4.31.15 IList.Add

Synopsis: Add an interface to the list.

Declaration: `function Add(item: IUnknown) : Integer`

Visibility: default

Description: `Add` adds the interface `Item` to the list, and returns the position at which it has been added.

See also: `TList.Add` (429), `IInterfaceList.Insert` (329), `IInterfaceList.Delete` (328)

4.31.16 IList.Insert

Synopsis: Insert an interface in the list.

Declaration: `procedure Insert(i: Integer; item: IUnknown)`

Visibility: default

Description: `Insert` inserts the interface `Item` in the list, at position `I`, shifting all items one position.

See also: `TList.Insert` (431), `IInterfaceList.Add` (329), `IInterfaceList.Delete` (328)

4.31.17 **IInterfaceList.Last**

Synopsis: Returns the last non-nil interface in the list.

Declaration: `function Last : IUnknown`

Visibility: default

Description: `Last` returns the last non-empty interface in the list.

See also: `TList.Last` ([432](#)), `IInterfaceList.First` ([329](#)), `IInterfaceList.IndexOf` ([329](#))

4.31.18 **IInterfaceList.Remove**

Synopsis: Remove an interface from the list.

Declaration: `function Remove(item: IUnknown) : Integer`

Visibility: default

Description: `Remove` searches for the first occurrence of `Item` in the list and deletes it.

See also: `TList.Remove` ([432](#)), `IInterfaceList.Delete` ([328](#)), `IInterfaceList.IndexOf` ([329](#))

4.31.19 **IInterfaceList.Lock**

Synopsis: Lock the list.

Declaration: `procedure Lock`

Visibility: default

Description: `Lock` locks the list. After a call to lock, the object list can only be accessed by the current thread, until `UnLock` ([330](#)) is called.

See also: `TInterfaceList.Lock` ([423](#)), `IInterfaceList.Unlock` ([330](#))

4.31.20 **IInterfaceList.Unlock**

Synopsis: Unlock the list.

Declaration: `procedure Unlock`

Visibility: default

Description: `Unlock` unlocks a locked list. After a call to `UnLock`, other threads are again able to access the list.

See also: `TInterfaceList.UnLock` ([423](#)), `IInterfaceList.Lock` ([330](#))

4.31.21 **IInterfaceList.Capacity**

Synopsis: Capacity of the list.

Declaration: `Property Capacity : Integer`

Visibility: default

Access: Read,Write

Description: `Capacity` is the maximum number of elements the list can hold without needing to reallocate memory for the list. It can be set to improve speed when adding a lot of items to the list.

See also: `TList.Capacity` ([433](#)), `IInterfaceList.Count` ([331](#))

4.31.22 IInterfaceList.Count

Synopsis: Current number of elements in the list.

Declaration: `Property Count : Integer`

Visibility: default

Access: Read,Write

Description: `Count` is the current number of elements in the list. Setting it to a larger number will allocate empty slots. Setting it to a smaller number will clear any interfaces that fall outside the new border.

See also: `IInterfaceList.Capacity` ([330](#)), `TList.Count` ([434](#))

4.31.23 IInterfaceList.Items

Synopsis: Provides Index-based, sequential, access to the interfaces in the list.

Declaration: `Property Items[index: Integer]: IUnknown; default`

Visibility: default

Access: Read,Write

Description: `Items` is the default property of the interface list and provides index-based array access to the interfaces in the list. Allowed values for `Index` include 0 to `Count-1`

See also: `IInterfaceList.Count` ([331](#)), `TList.Items` ([434](#))

4.32 IInterfaceListEx

4.32.1 Method overview

Page	Method	Description
331	<code>GetEnumerator</code>	

4.32.2 IInterfaceListEx.GetEnumerator

Declaration: `function GetEnumerator : TInterfaceListEnumerator`

Visibility: default

4.33 IStreamPersist

4.33.1 Description

`IStreamPersist` defines an interface for object persistence streaming to a stream. Any class implementing this interface is expected to be able to save or load it's state from or to a stream.

See also: `TPersistent` ([446](#)), `TComponent` ([386](#)), `TStream` ([466](#))

4.33.2 Method overview

Page	Method	Description
332	<code>LoadFromStream</code>	Load persistent data from stream.
332	<code>SaveToStream</code>	Save persistent data to stream.

4.33.3 `IStreamPersist.LoadFromStream`

Synopsis: Load persistent data from stream.

Declaration: `procedure LoadFromStream(Stream: TStream)`

Visibility: default

Description: `LoadFromStream` is the method called when the object should load it's state from the stream `stream`. It should be able to read the data which was written using the `SaveToStream` method.

See also: `TPersistent` ([446](#)), `TComponent` ([386](#)), `TStream` ([466](#)), `IStreamPersist.SaveToStream` ([332](#))

4.33.4 `IStreamPersist.SaveToStream`

Synopsis: Save persistent data to stream.

Declaration: `procedure SaveToStream(Stream: TStream)`

Visibility: default

Description: `SaveFromStream` is the method called when the object should load it's state from the stream `stream`. The data written by this method should be readable by the `LoadFromStream` method.

See also: `TPersistent` ([446](#)), `TComponent` ([386](#)), `TStream` ([466](#)), `IStreamPersist.LoadFromStream` ([332](#))

4.34 `IStringsAdapter`

4.34.1 Description

Is not yet supported in Free Pascal.

See also: `TStrings` ([486](#))

4.34.2 Method overview

Page	Method	Description
332	<code>ReferenceStrings</code>	Add a reference to the indicated strings.
333	<code>ReleaseStrings</code>	Release the reference to the strings.

4.34.3 `IStringsAdapter.ReferenceStrings`

Synopsis: Add a reference to the indicated strings.

Declaration: `procedure ReferenceStrings(S: TStrings)`

Visibility: default

4.34.4 IStringsAdapter.ReleaseStrings

Synopsis: Release the reference to the strings.

Declaration: `procedure ReleaseStrings`

Visibility: `default`

4.35 IVCLComObject

4.35.1 Description

`IVCLComObject` is used by `TComponent` to implement the `IUnknown` interface used by COM automation servers. Partially, it is the translation to pascal of the `IDispatch` interface definition by Microsoft. If `TComponent` needs to return an `IUnknown` interface, it creates a `IVCLComObject` interface instead.

See also: `TComponent.VCLComObject` ([395](#))

4.35.2 Method overview

Page	Method	Description
335	<code>FreeOnRelease</code>	Is called by <code>TComponent.FreeOnRelease</code> .
334	<code>GetIDsOfNames</code>	The <code>IDispatch: GetIDsOfNames</code> call for automation servers.
333	<code>GetTypeInfo</code>	The <code>IDispatch: GetTypeInfo</code> call for automation servers.
333	<code>GetTypeInfoCount</code>	The <code>IDispatch: GetTypeInfoCount</code> call for automation servers.
334	<code>Invoke</code>	The <code>IDispatch: Invoke</code> call for automation servers.
334	<code>SafeCallException</code>	This method can be invoked if an exception occurs during <code>Invoke</code> .

4.35.3 IVCLComObject.GetTypeInfoCount

Synopsis: The `IDispatch: GetTypeInfoCount` call for automation servers.

Declaration: `function GetTypeInfoCount(out Count: Integer) : HRESULT`

Visibility: `default`

Description: `GetTypeInfoCount` must return in `Count` either 0 or 1 to indicate that it provides type information (1) or not (0).

Errors: On error, a nonzero (different from `S_OK`) return value must be returned.

See also: `IVCLComObject.GetTypeInfo` ([333](#))

4.35.4 IVCLComObject.GetTypeInfo

Synopsis: The `IDispatch: GetTypeInfo` call for automation servers.

Declaration: `function GetTypeInfo(Index: Integer; LocaleID: Integer; out TypeInfo) : HRESULT`

Visibility: `default`

Description: `GetTypeInfo` must return the `Index`-th entry in the type information of the component in `TypeInfo`. The `LocaleID` argument can be used to indicate the locale of the caller, as different type information can be returned depending on the locale.

Errors: On error, a nonzero (different from S_OK) return value must be returned.

See also: `IVCLComObject.GetTypeInfoCount` ([333](#))

4.35.5 `IVCLComObject.GetIDsOfNames`

Synopsis: The `IDispatch::GetIDsOfNames` call for automation servers.

Declaration: `function GetIDsOfNames(const IID: TGuid; Names: Pointer;
NameCount: Integer; LocaleID: Integer;
DispIDs: Pointer) : HRESULT`

Visibility: default

Description: `GetIDsOfNames` must return in `DispIDs` the dispatch Ids for the `NameCount` names of the methods listed in `Names`. The `LocaleID` indicates the locale of the caller.

Errors: On error, a nonzero (different from S_OK) return value must be returned.

See also: `IVCLComObject.Invoke` ([334](#))

4.35.6 `IVCLComObject.Invoke`

Synopsis: The `IDispatch::Invoke` call for automation servers.

Declaration: `function Invoke(DispID: Integer; const IID: TGuid; LocaleID: Integer;
Flags: Word; var Params; VarResult: Pointer;
ExcepInfo: Pointer; ArgErr: Pointer) : HRESULT`

Visibility: default

Description: `Invoke` must invoke the method designated by `DispID`. `IID` can be ignored. `LocaleID` is used by the caller to indicate the locale it is using. The `Flags` argument describes the context in which `Invoke` is called: a method, or property getter/setter. The `Params` argument contains the parameters to the call. The result should be in `VarResult`. On error, `ExcepInfo` and `ArgError` should be filled.

The function should return 0 (S_OK) if all went well.

See also: `IVCLComObject.GetIDsOfNames` ([334](#))

4.35.7 `IVCLComObject.SafeCallException`

Synopsis: This method can be invoked if an exception occurs during `Invoke`.

Declaration: `function SafeCallException(ExceptObject: TObject;
ExceptAddr: CodePointer) : HRESULT`

Visibility: default

Description: `SafeCallException` is called to handle an exception during invocation of the `Invoke` method. The `TObject` implementation of this method returns `E_UNEXPECTED`.

See also: `IVCLComObject.Invoke` ([334](#))

4.35.8 IVCLComObject.FreeOnRelease

Synopsis: Is called by TComponent.FreeOnRelease.

Declaration: `procedure FreeOnRelease`

Visibility: `default`

Description: `FreeOnRelease` is called by `TComponent.FreeOnRelease` ([390](#)) for the `IVCLComObject` interface implemented by `TComponent`.

See also: `TComponent.FreeOnRelease` ([390](#))

4.36 TAbstractObjectReader

4.36.1 Description

The Free Pascal streaming mechanism, while compatible with Delphi's mechanism, differs from it in the sense that the streaming mechanism uses a driver class when streaming components. The `TAbstractObjectReader` class is the base driver class for reading property values from streams. It consists entirely of abstract methods, which must be implemented by descendant classes.

Different streaming mechanisms can be implemented by making a descendant from `TAbstractObjectReader`. The `TBinaryObjectReader` ([356](#)) class is such a descendant class, which streams data in binary (Delphi compatible) format.

All methods described in this class, must be implemented by descendant classes.

See also: `TBinaryObjectReader` ([356](#))

4.36.2 Method overview

Page	Method	Description
337	<code>BeginComponent</code>	Marks the reading of a new component.
337	<code>BeginProperty</code>	Marks the reading of a property value.
337	<code>BeginRootComponent</code>	Starts the reading of the root component.
336	<code>FlushBuffer</code>	Flush the buffer.
336	<code>NextValue</code>	Returns the type of the next value in the stream.
338	<code>Read</code>	Read raw data from stream.
338	<code>ReadBinary</code>	Read binary data from the stream.
339	<code>ReadCurrency</code>	Read a currency value from the stream.
339	<code>ReadDate</code>	Read a date value from the stream.
339	<code>ReadDouble</code>	
338	<code>ReadFloat</code>	Read a float value from the stream.
339	<code>ReadIdent</code>	Read an identifier from the stream.
340	<code>ReadInt16</code>	Read a 16-bit integer from the stream.
341	<code>ReadInt32</code>	Read a 32-bit integer from the stream.
341	<code>ReadInt64</code>	Read a 64-bit integer from the stream.
340	<code>ReadInt8</code>	Read an 8-bit integer from the stream.
341	<code>ReadSet</code>	Reads a set from the stream.
342	<code>ReadSignature</code>	Read resource signature.
338	<code>ReadSingle</code>	Read a single (real-type) value from the stream.
342	<code>ReadStr</code>	Read a shortstring from the stream.
342	<code>ReadString</code>	Read a string of type <code>StringType</code> from the stream.
343	<code>ReadUnicodeString</code>	Read a Unicode string value.
337	<code>ReadValue</code>	Reads the type of the next value.
343	<code>ReadWideString</code>	Read a widestring value from the stream.
343	<code>SkipComponent</code>	Skip till the end of the component.
343	<code>SkipValue</code>	Skip the current value.

4.36.3 TAbstractObjectReader.FlushBuffer

Synopsis: Flush the buffer.

Declaration: `procedure FlushBuffer; Virtual`

Visibility: `public`

Description: `FlushBuffer` flushes the buffer. It is provided for Delphi compatibility, and is not used in FPC.

See also: `TFile.FlushBuffer` ([404](#))

4.36.4 TAbstractObjectReader.NextValue

Synopsis: Returns the type of the next value in the stream.

Declaration: `function NextValue : TValueType; Virtual; Abstract`

Visibility: `public`

Description: This function should return the type of the next value in the stream, but should not read the actual value, i.e. the stream position should not be altered by this method. This is used to 'peek' in the stream what value is next.

See also: `TAbstractObjectReader.ReadValue` ([337](#))

4.36.5 TAbstractObjectReader.ReadValue

Synopsis: Reads the type of the next value.

Declaration: `function ReadValue : TValueType; Virtual; Abstract`

Visibility: public

Description: This function returns the type of the next value in the stream and reads it. i.e. after the call to this method, the stream is positioned to read the value of the type returned by this function.

See also: `TAbstractObjectReader.ReadValue` ([337](#))

4.36.6 TAbstractObjectReader.BeginRootComponent

Synopsis: Starts the reading of the root component.

Declaration: `procedure BeginRootComponent; Virtual; Abstract`

Visibility: public

Description: This function can be used to initialize the driver class for reading a component. It is called once at the beginning of the read process, and is immediately followed by a call to `BeginComponent` ([337](#)).

See also: `TAbstractObjectReader.BeginComponent` ([337](#))

4.36.7 TAbstractObjectReader.BeginComponent

Synopsis: Marks the reading of a new component.

Declaration: `procedure BeginComponent (var Flags: TFileFlags;
var AChildPos: Integer;
var CompClassName: string; var CompName: string)
; Virtual; Abstract`

Visibility: public

Description: This method is called when the streaming process wants to start reading a new component.

Descendant classes should override this method to read the start of a component new component definition and return the needed arguments. `Flags` should be filled with any flags that were found at the component definition, as well as `AChildPos`. The `CompClassName` should be filled with the class name of the streamed component, and the `CompName` argument should be filled with the name of the component.

`AChildPos` is used to change the ordering in which components appear below their parent component when streaming descendant forms.

See also: `TAbstractObjectReader.BeginRootComponent` ([337](#)), `TAbstractObjectReader.BeginProperty` ([337](#))

4.36.8 TAbstractObjectReader.BeginProperty

Synopsis: Marks the reading of a property value.

Declaration: `function BeginProperty : string; Virtual; Abstract`

Visibility: public

Description: `BeginProperty` is called by the streaming system when it wants to read a new property. The return value of the function is the name of the property which can be read from the stream.

See also: `TAbstractObjectReader.BeginComponent` ([337](#))

4.36.9 TAbstractObjectReader.Read

Synopsis: Read raw data from stream.

Declaration: `procedure Read(var Buf; Count: LongInt); Virtual; Abstract`

Visibility: public

Description: `Read` is introduced for Delphi compatibility to read raw data from the component stream. This should not be used in production code as it will totally mess up the streaming.

See also: `TBinaryObjectReader.Read` (358), `TReader.Read` (455)

4.36.10 TAbstractObjectReader.ReadBinary

Synopsis: Read binary data from the stream.

Declaration: `procedure ReadBinary(const DestData: TMemoryStream); Virtual; Abstract`

Visibility: public

Description: `ReadBinary` is called when binary data should be read from the stream (i.e. after `ReadValue` (337) returned a value type of `vaBinary`). The data should be stored in the `DestData` memory stream by descendant classes.

See also: `TAbstractObjectReader.ReadFloat` (338), `TAbstractObjectReader.ReadDate` (339), `TAbstractObjectReader.ReadSingle` (338), `TAbstractObjectReader.ReadIdent` (339), `TAbstractObjectReader.ReadInt8` (340), `TAbstractObjectReader.ReadInt16` (340), `TAbstractObjectReader.ReadInt32` (341), `TAbstractObjectReader.ReadInt64` (341), `TabstractObjectReader.ReadSet` (341), `TabstractObjectReader.ReadStr` (342), `TabstractObjectReader.ReadString` (342)

4.36.11 TAbstractObjectReader.ReadFloat

Synopsis: Read a float value from the stream.

Declaration: `function ReadFloat : Extended; Virtual; Abstract`

Visibility: public

Description: `ReadFloat` is called by the streaming system when it wants to read a float from the stream (i.e. after `ReadValue` (337) returned a value type of `vaExtended`). The return value should be the value of the float.

See also: `TAbstractObjectReader.ReadFloat` (338), `TAbstractObjectReader.ReadDate` (339), `TAbstractObjectReader.ReadSingle` (338), `TAbstractObjectReader.ReadIdent` (339), `TAbstractObjectReader.ReadInt8` (340), `TAbstractObjectReader.ReadInt16` (340), `TAbstractObjectReader.ReadInt32` (341), `TAbstractObjectReader.ReadInt64` (341), `TabstractObjectReader.ReadSet` (341), `TabstractObjectReader.ReadStr` (342), `TabstractObjectReader.ReadString` (342)

4.36.12 TAbstractObjectReader.ReadSingle

Synopsis: Read a single (real-type) value from the stream.

Declaration: `function ReadSingle : Single; Virtual; Abstract`

Visibility: public

Description: `ReadSingle` is called by the streaming system when it wants to read a single-type float from the stream (i.e. after `ReadValue` (337) returned a value type of `vaSingle`). The return value should be the value of the float.

See also: `TAbstractObjectReader.ReadFloat` (338), `TAbstractObjectReader.ReadDate` (339), `TAbstractObjectReader.ReadSingle` (338), `TAbstractObjectReader.ReadIdent` (339), `TAbstractObjectReader.ReadInt8` (340), `TAbstractObjectReader.ReadInt16` (340), `TAbstractObjectReader.ReadInt32` (341), `TAbstractObjectReader.ReadInt64` (341), `TabstractObjectReader.ReadSet` (341), `TabstractObjectReader.ReadStr` (342), `TabstractObjectReader.ReadString` (342)

4.36.13 TAbstractObjectReader.ReadDouble

Declaration: `function ReadDouble : Double; Virtual; Abstract`

Visibility: public

4.36.14 TAbstractObjectReader.ReadDate

Synopsis: Read a date value from the stream.

Declaration: `function ReadDate : TDateTime; Virtual; Abstract`

Visibility: public

Description: `ReadDate` is called by the streaming system when it wants to read a date/time value from the stream (i.e. after `ReadValue` (337) returned a value type of `vaDate`). The return value should be the date/time value. (This value can be stored as a float, since `TDateTime` is nothing but a float.)

See also: `TAbstractObjectReader.ReadFloat` (338), `TAbstractObjectReader.ReadSingle` (338), `TAbstractObjectReader.ReadIdent` (339), `TAbstractObjectReader.ReadInt8` (340), `TAbstractObjectReader.ReadInt16` (340), `TAbstractObjectReader.ReadInt32` (341), `TAbstractObjectReader.ReadInt64` (341), `TabstractObjectReader.ReadSet` (341), `TabstractObjectReader.ReadStr` (342), `TabstractObjectReader.ReadString` (342)

4.36.15 TAbstractObjectReader.ReadCurrency

Synopsis: Read a currency value from the stream.

Declaration: `function ReadCurrency : Currency; Virtual; Abstract`

Visibility: public

Description: `ReadCurrency` is called when a currency-typed value should be read from the stream. This abstract method should be overridden by descendant classes, and should return the currency value read from the stream.

See also: `TAbstractObjectWriter.WriteCurrency` (347)

4.36.16 TAbstractObjectReader.ReadIdent

Synopsis: Read an identifier from the stream.

Declaration: `function ReadIdent (ValueType: TValueType) : string; Virtual; Abstract`

Visibility: public

Description: `ReadIdent` is called by the streaming system if it expects to read an identifier of type `ValueType` from the stream after a call to `ReadValue` (337) returned `vaIdent`. The identifier should be returned as a string. Note that in some cases the identifier does not actually have to be in the stream. The following table indicates which identifiers must actually be read:

Table 4.24:

ValueType	Expected value
<code>vaIdent</code>	Read from stream.
<code>vaNil</code>	'Nil'. This does not have to be read from the stream.
<code>vaFalse</code>	'False'. This does not have to be read from the stream.
<code>vaTrue</code>	'True'. This does not have to be read from the stream.
<code>vaNull</code>	'Null'. This does not have to be read from the stream.

See also: `TAbstractObjectReader.ReadFloat` (338), `TAbstractObjectReader.ReadDate` (339), `TAbstractObjectReader.ReadSingle` (338), `TAbstractObjectReader.ReadInt8` (340), `TAbstractObjectReader.ReadInt16` (340), `TAbstractObjectReader.ReadInt32` (341), `TAbstractObjectReader.ReadInt64` (341), `TAbstractObjectReader.ReadSet` (341), `TAbstractObjectReader.ReadStr` (342), `TAbstractObjectReader.ReadString` (342)

4.36.17 TAbstractObjectReader.ReadInt8

Synopsis: Read an 8-bit integer from the stream.

Declaration: `function ReadInt8 : ShortInt; Virtual; Abstract`

Visibility: `public`

Description: `ReadInt8` is called by the streaming process if it expects to read an integer value with a size of 8 bits (1 byte) from the stream (i.e. after `ReadValue` (337) returned a `valuetype` of `vaInt8`). The return value is the value of the integer. Note that the size of the value in the stream does not actually have to be 1 byte.

See also: `TAbstractObjectReader.ReadFloat` (338), `TAbstractObjectReader.ReadDate` (339), `TAbstractObjectReader.ReadSingle` (338), `TAbstractObjectReader.ReadIdent` (339), `TAbstractObjectReader.ReadInt16` (340), `TAbstractObjectReader.ReadInt32` (341), `TAbstractObjectReader.ReadInt64` (341), `TAbstractObjectReader.ReadSet` (341), `TAbstractObjectReader.ReadStr` (342), `TAbstractObjectReader.ReadString` (342)

4.36.18 TAbstractObjectReader.ReadInt16

Synopsis: Read a 16-bit integer from the stream.

Declaration: `function ReadInt16 : SmallInt; Virtual; Abstract`

Visibility: `public`

Description: `ReadInt16` is called by the streaming process if it expects to read an integer value with a size of 16 bits (2 bytes) from the stream (i.e. after `ReadValue` (337) returned a `valuetype` of `vaInt16`). The return value is the value of the integer. Note that the size of the value in the stream does not actually have to be 2 bytes.

See also: `TAbstractObjectReader.ReadFloat` (338), `TAbstractObjectReader.ReadDate` (339), `TAbstractObjectReader.ReadSingle` (338), `TAbstractObjectReader.ReadIdent` (339), `TAbstractObjectReader.ReadInt8` (340), `TAbstractObjectReader.ReadInt32` (341), `TAbstractObjectReader.ReadInt64` (341), `TAbstractObjectReader.ReadSet` (341), `TAbstractObjectReader.ReadStr` (342), `TAbstractObjectReader.ReadString` (342)

4.36.19 `TAbstractObjectReader.ReadInt32`

Synopsis: Read a 32-bit integer from the stream.

Declaration: `function ReadInt32 : LongInt; Virtual; Abstract`

Visibility: public

Description: `ReadInt32` is called by the streaming process if it expects to read an integer value with a size of 32 bits (4 bytes) from the stream (i.e. after `ReadValue` (337) returned a valuetype of `vaInt32`). The return value is the value of the integer. Note that the size of the value in the stream does not actually have to be 4 bytes.

See also: `TAbstractObjectReader.ReadFloat` (338), `TAbstractObjectReader.ReadDate` (339), `TAbstractObjectReader.ReadSingle` (338), `TAbstractObjectReader.ReadIdent` (339), `TAbstractObjectReader.ReadInt8` (340), `TAbstractObjectReader.ReadInt16` (340), `TAbstractObjectReader.ReadInt64` (341), `TAbstractObjectReader.ReadSet` (341), `TAbstractObjectReader.ReadStr` (342), `TAbstractObjectReader.ReadString` (342)

4.36.20 `TAbstractObjectReader.ReadInt64`

Synopsis: Read a 64-bit integer from the stream.

Declaration: `function ReadInt64 : Int64; Virtual; Abstract`

Visibility: public

Description: `ReadInt64` is called by the streaming process if it expects to read an `int64` value with a size of 64 bits (8 bytes) from the stream (i.e. after `ReadValue` (337) returned a valuetype of `vaInt64`). The return value is the value of the integer. Note that the size of the value in the stream does not actually have to be 8 bytes.

See also: `TAbstractObjectReader.ReadFloat` (338), `TAbstractObjectReader.ReadDate` (339), `TAbstractObjectReader.ReadSingle` (338), `TAbstractObjectReader.ReadIdent` (339), `TAbstractObjectReader.ReadInt8` (340), `TAbstractObjectReader.ReadInt16` (340), `TAbstractObjectReader.ReadInt32` (341), `TAbstractObjectReader.ReadSet` (341), `TAbstractObjectReader.ReadStr` (342), `TAbstractObjectReader.ReadString` (342)

4.36.21 `TAbstractObjectReader.ReadSet`

Synopsis: Reads a set from the stream.

Declaration: `function ReadSet (EnumType: Pointer) : Integer; Virtual; Abstract`

Visibility: public

Description: This method is called by the streaming system if it expects to read a set from the stream (i.e. after `ReadValue` (337) returned a valuetype of `vaSet`). The return value is the contents of the set, encoded in a bitmask the following way:

For each (enumerated) value in the set, the bit corresponding to the ordinal value of the enumerated value should be set. i.e. as `1 shl ord(value)`.

See also: `TAbstractObjectReader.ReadFloat` (338), `TAbstractObjectReader.ReadDate` (339), `TAbstractObjectReader.ReadSingle` (338), `TAbstractObjectReader.ReadIdent` (339), `TAbstractObjectReader.ReadInt8` (340), `TAbstractObjectReader.ReadInt16` (340), `TAbstractObjectReader.ReadInt32` (341), `TAbstractObjectReader.ReadInt64` (341), `TAbstractObjectReader.ReadStr` (342), `TAbstractObjectReader.ReadString` (342)

4.36.22 TAbstractObjectReader.ReadSignature

Synopsis: Read resource signature.

Declaration: `procedure ReadSignature; Virtual; Abstract`

Visibility: public

Description: `ReadSignature` reads the streaming signature from a stream. This method does nothing. It must be implemented by descendents that have a signature header in the stream. (such as binary streams)

See also: `TAbstractObjectWriter.WriteSignature` (345), `TBinaryObjectReader.ReadSignature` (361)

4.36.23 TAbstractObjectReader.ReadStr

Synopsis: Read a shortstring from the stream.

Declaration: `function ReadStr : string; Virtual; Abstract`

Visibility: public

Description: `ReadStr` is called by the streaming system if it expects to read a string from the stream (i.e. after `ReadValue` (337) returned a valuetype of `vaLString`, `vaWstring` or `vaString`). The return value is the string.

See also: `TAbstractObjectReader.ReadFloat` (338), `TAbstractObjectReader.ReadDate` (339), `TAbstractObjectReader.ReadSingle` (338), `TAbstractObjectReader.ReadIdent` (339), `TAbstractObjectReader.ReadInt8` (340), `TAbstractObjectReader.ReadInt16` (340), `TAbstractObjectReader.ReadInt32` (341), `TAbstractObjectReader.ReadInt64` (341), `TAbstractObjectReader.ReadSet` (341), `TAbstractObjectReader.ReadStr` (342)

4.36.24 TAbstractObjectReader.ReadString

Synopsis: Read a string of type `StringType` from the stream.

Declaration: `function ReadString(StringType: TValueType) : string; Virtual; Abstract`

Visibility: public

Description: `ReadStr` is called by the streaming system if it expects to read a string from the stream (i.e. after `ReadValue` (337) returned a valuetype of `vaLString`, `vaWstring` or `vaString`). The return value is the string.

See also: `TAbstractObjectReader.ReadFloat` (338), `TAbstractObjectReader.ReadDate` (339), `TAbstractObjectReader.ReadSingle` (338), `TAbstractObjectReader.ReadIdent` (339), `TAbstractObjectReader.ReadInt8` (340), `TAbstractObjectReader.ReadInt16` (340), `TAbstractObjectReader.ReadInt32` (341), `TAbstractObjectReader.ReadInt64` (341), `TAbstractObjectReader.ReadSet` (341), `TAbstractObjectReader.ReadStr` (342)

4.36.25 TAbstractObjectReader.ReadWideString

Synopsis: Read a widestring value from the stream.

Declaration: `function ReadWideString : WideString; Virtual; Abstract`

Visibility: public

Description: `ReadWideString` is called when a widestring-typed value should be read from the stream. This abstract method should be overridden by descendant classes.

See also: `TAbstractObjectWriter.WriteWideString` ([349](#))

4.36.26 TAbstractObjectReader.ReadUnicodeString

Synopsis: Read a Unicode string value.

Declaration: `function ReadUnicodeString : UnicodeString; Virtual; Abstract`

Visibility: public

Description: `ReadUnicodeString` should read a `UnicodeString` value from the stream. (indicated by the `vaUString` value type).

Descendant classes should override this method to actually read a `UnicodeString` value.

See also: `TBinaryObjectWriter.WriteUnicodeString` ([367](#)), `TAbstractObjectReader.ReadWideString` ([343](#))

4.36.27 TAbstractObjectReader.SkipComponent

Synopsis: Skip till the end of the component.

Declaration: `procedure SkipComponent (SkipComponentInfos: Boolean); Virtual
; Abstract`

Visibility: public

Description: This method is used to skip the entire declaration of a component in the stream. Each descendant of `TAbstractObjectReader` should implement this in a way which is optimal for the implemented stream format.

See also: `TAbstractObjectReader.BeginComponent` ([337](#)), `TAbstractObjectReader.SkipValue` ([343](#))

4.36.28 TAbstractObjectReader.SkipValue

Synopsis: Skip the current value.

Declaration: `procedure SkipValue; Virtual; Abstract`

Visibility: public

Description: `SkipValue` should be used when skipping a value in the stream; The method should determine the type of the value which should be skipped by itself, if this is necessary.

See also: `TAbstractObjectReader.SkipComponent` ([343](#))

4.37 TAbstractObjectWriter

4.37.1 Description

Abstract driver class for writing component data.

4.37.2 Method overview

Page	Method	Description
344	BeginCollection	Start writing a collection.
344	BeginComponent	Start writing a component.
345	BeginList	Start writing a list.
345	BeginProperty	Start writing a property.
345	EndList	Mark the end of a list.
345	EndProperty	Marks the end of writing of a property.
346	FlushBuffer	Flush the buffer
346	Write	Write raw data to stream.
346	WriteBinary	Writes binary data to the stream.
346	WriteBoolean	Writes a boolean value to the stream.
347	WriteCurrency	Write a currency value to the stream.
347	WriteDate	Writes a date type to the stream.
346	WriteFloat	Writes a float value to the stream.
347	WriteIdent	Writes an identifier to the stream.
347	WriteInteger	Writes an integer value to the stream.
348	WriteMethodName	Writes a methodname to the stream.
348	WriteSet	Writes a set value to the stream.
345	WriteSignature	Write stream signature to the stream.
347	WriteSingle	Writes a single-type real value to the stream.
348	WriteString	Writes a string value to the stream.
348	WriteUInt64	Write an unsigned 64-bit integer.
349	WriteUnicodeString	Write a Unicode string to the stream.
348	WriteVariant	Write a variant to the stream.
349	WriteWideString	Write a widestring value to the stream.

4.37.3 TAbstractObjectWriter.BeginCollection

Synopsis: Start writing a collection.

Declaration: `procedure BeginCollection; Virtual; Abstract`

Visibility: `public`

Description: Start writing a collection.

4.37.4 TAbstractObjectWriter.BeginComponent

Synopsis: Start writing a component.

Declaration: `procedure BeginComponent (Component: TComponent; Flags: TFileFlags;
ChildPos: Integer); Virtual; Abstract`

Visibility: `public`

Description: Start writing a component.

4.37.5 TAbstractObjectWriter.WriteSignature

Synopsis: Write stream signature to the stream.

Declaration: `procedure WriteSignature; Virtual; Abstract`

Visibility: `public`

Description: `WriteSignature` writes the streaming signature to a stream. This method does nothing, it must be implemented by descendents that have a signature header in the stream. (such as binary streams)

See also: `TAbstractObjectReader.ReadSignature` ([342](#)), `TBinaryObjectWriter.WriteSignature` ([364](#))

4.37.6 TAbstractObjectWriter.BeginList

Synopsis: Start writing a list.

Declaration: `procedure BeginList; Virtual; Abstract`

Visibility: `public`

Description: Start writing a list.

4.37.7 TAbstractObjectWriter.EndList

Synopsis: Mark the end of a list.

Declaration: `procedure EndList; Virtual; Abstract`

Visibility: `public`

Description: Mark the end of a list.

4.37.8 TAbstractObjectWriter.BeginProperty

Synopsis: Start writing a property.

Declaration: `procedure BeginProperty(const PropName: string); Virtual; Abstract`

Visibility: `public`

Description: Start writing a property.

4.37.9 TAbstractObjectWriter.EndProperty

Synopsis: Marks the end of writing of a property.

Declaration: `procedure EndProperty; Virtual; Abstract`

Visibility: `public`

Description: Marks the end of writing of a property.

4.37.10 TAbstractObjectWriter.FlushBuffer

Synopsis: Flush the buffer

Declaration: `procedure FlushBuffer; Virtual`

Visibility: `public`

Description: `FlushBuffer` flushes the buffer. It is provided for Delphi compatibility, and is not used in FPC.

See also: `TFile.FlushBuffer` ([404](#)), `TWriter.FlushBuffer` ([538](#))

4.37.11 TAbstractObjectWriter.Write

Synopsis: Write raw data to stream.

Declaration: `procedure Write(const Buffer; Count: LongInt); Virtual; Abstract`

Visibility: `public`

Description: `Write` is introduced for Delphi compatibility to write raw data to the component stream. This should not be used in new production code as it will totally mess up the streaming.

See also: `TBinaryObjectWriter.Write` ([365](#)), `TWriter.Write` ([539](#))

4.37.12 TAbstractObjectWriter.WriteBinary

Synopsis: Writes binary data to the stream.

Declaration: `procedure WriteBinary(const Buffer; Count: LongInt); Virtual; Abstract`

Visibility: `public`

Description: Writes binary data to the stream.

4.37.13 TAbstractObjectWriter.WriteBoolean

Synopsis: Writes a boolean value to the stream.

Declaration: `procedure WriteBoolean(Value: Boolean); Virtual; Abstract`

Visibility: `public`

Description: Writes a boolean value to the stream.

4.37.14 TAbstractObjectWriter.WriteFloat

Synopsis: Writes a float value to the stream.

Declaration: `procedure WriteFloat(const Value: Extended); Virtual; Abstract`

Visibility: `public`

Description: Writes a float value to the stream.

4.37.15 TAbstractObjectWriter.WriteSingle

Synopsis: Writes a single-type real value to the stream.

Declaration: `procedure WriteSingle(const Value: Single); Virtual; Abstract`

Visibility: `public`

Description: Writes a single-type real value to the stream.

4.37.16 TAbstractObjectWriter.WriteDate

Synopsis: Writes a date type to the stream.

Declaration: `procedure WriteDate(const Value: TDateTime); Virtual; Abstract`

Visibility: `public`

Description: Writes a date type to the stream.

4.37.17 TAbstractObjectWriter.WriteCurrency

Synopsis: Write a currency value to the stream.

Declaration: `procedure WriteCurrency(const Value: Currency); Virtual; Abstract`

Visibility: `public`

Description: `WriteCurrency` is called when a currency-typed value should be written to the stream. This abstract method should be overridden by descendant classes.

See also: `TAbstractObjectReader.ReadCurrency` ([339](#))

4.37.18 TAbstractObjectWriter.WriteIdent

Synopsis: Writes an identifier to the stream.

Declaration: `procedure WriteIdent(const Ident: string); Virtual; Abstract`

Visibility: `public`

Description: Writes an identifier to the stream.

4.37.19 TAbstractObjectWriter.WriteInteger

Synopsis: Writes an integer value to the stream.

Declaration: `procedure WriteInteger(Value: Int64); Virtual; Abstract`

Visibility: `public`

Description: Writes an integer value to the stream.

4.37.20 TAbstractObjectWriter.WriteUInt64

Synopsis: Write an unsigned 64-bit integer.

Declaration: `procedure WriteUInt64(Value: QWord); Virtual; Abstract`

Visibility: public

Description: `WriteUInt64` must be overridden by descendant classes to write a 64-bit unsigned Value (value type `QWord`) to the stream.

Errors: None.

See also: `TBinaryObjectWriter.WriteUInt64` ([366](#))

4.37.21 TAbstractObjectWriter.WriteVariant

Synopsis: Write a variant to the stream.

Declaration: `procedure WriteVariant(const Value: Variant); Virtual; Abstract`

Visibility: public

Description: `WriteVariant` must be overridden by descendant classes to write a simple variant type to the stream. `WriteVariant` does not write arrays types or complex types.

See also: `TBinaryObjectWriter.WriteVariant` ([368](#))

4.37.22 TAbstractObjectWriter.WriteMethodName

Synopsis: Writes a methodname to the stream.

Declaration: `procedure WriteMethodName(const Name: string); Virtual; Abstract`

Visibility: public

Description: Writes a methodname to the stream.

4.37.23 TAbstractObjectWriter.WriteSet

Synopsis: Writes a set value to the stream.

Declaration: `procedure WriteSet(Value: LongInt; SetType: Pointer); Virtual
; Abstract`

Visibility: public

Description: Writes a set value to the stream.

4.37.24 TAbstractObjectWriter.WriteString

Synopsis: Writes a string value to the stream.

Declaration: `procedure WriteString(const Value: string); Virtual; Abstract`

Visibility: public

Description: Writes a string value to the stream.

4.37.25 TAbstractObjectWriter.WriteWideString

Synopsis: Write a widestring value to the stream.

Declaration: `procedure WriteWideString(const Value: WideString); Virtual; Abstract`

Visibility: public

Description: `WriteCurrency` is called when a currency-typed value should be written to the stream. This abstract method should be overridden by descendant classes.

See also: `TAbstractObjectReader.ReadWideString` (343)

4.37.26 TAbstractObjectWriter.WriteUnicodeString

Synopsis: Write a Unicode string to the stream.

Declaration: `procedure WriteUnicodeString(const Value: UnicodeString); Virtual; Abstract`

Visibility: public

Description: `WriteUnicodeString` must be overridden by descendant classes to write a unicodestring (value type `vaUString`) value to the stream.

See also: `TBinaryObjectWriter.WriteUnicodeString` (367)

4.38 TBasicAction

4.38.1 Description

`TBasicAction` implements a basic action class from which all actions are derived. It introduces all basic methods of an action, and implements functionality to maintain a list of clients, i.e. components that are connected with this action.

Do not create instances of `TBasicAction`. Instead, create a descendant class and create an instance of this class instead.

See also: `TBasicActionLink` (353), `TComponent` (386)

4.38.2 Method overview

Page	Method	Description
350	Create	Creates a new instance of a <code>TBasicAction</code> (349) class.
350	Destroy	Destroys the action.
351	Execute	Triggers the <code>OnExecute</code> (353) event.
351	ExecuteTarget	Executes the action on the <code>Target</code> object.
350	HandlesTarget	Determines whether <code>Target</code> can be handled by this action.
351	RegisterChanges	Registers a new client with the action.
352	Suspended	
352	UnRegisterChanges	Unregisters a client from the list of clients.
352	Update	Triggers the <code>OnUpdate</code> (353) event.
351	UpdateTarget	Notify client controls when the action updates itself.

4.38.3 Property overview

Page	Properties	Access	Description
352	ActionComponent	rw	Returns the component that initiated the action.
353	Clients	r	
353	OnExecute	rw	Event triggered when the action executes.
353	OnUpdate	rw	Event triggered when the application is idle.

4.38.4 TBasicAction.Create

Synopsis: Creates a new instance of a `TBasicAction` ([349](#)) class.

Declaration: `constructor Create(AOwner: TComponent); Override`

Visibility: `public`

Description: `Create` calls the inherited constructor, and then initializes the list of clients controls (or action lists).

Under normal circumstances it should not be necessary to create a `TBasicAction` descendant manually, actions are created in an IDE.

See also: `Destroy` ([350](#)), `AssignClient` ([349](#))

4.38.5 TBasicAction.Destroy

Synopsis: Destroys the action.

Declaration: `destructor Destroy; Override`

Visibility: `public`

Description: `Destroy` cleans up the list of client controls and then calls the inherited destructor.

An application programmer should not call `Destroy` directly; Instead `Free` should be called, if it needs to be called at all. Normally the controlling class (e.g. a `TActionList`) will destroy the action.

4.38.6 TBasicAction.HandlesTarget

Synopsis: Determines whether `Target` can be handled by this action.

Declaration: `function HandlesTarget(Target: TObject) : Boolean; Virtual`

Visibility: `public`

Description: `HandlesTarget` returns `True` if `Target` is a valid client for this action and if so, if it is in a suitable state to execute the action. An application programmer should never need to call `HandlesTarget` directly, it will be called by the action itself when needed.

In `TBasicAction` this method is empty; descendant classes should override this method to implement appropriate checks.

See also: `UpdateTarget` ([351](#)), `ExecuteTarget` ([351](#))

4.38.7 TBasicAction.UpdateTarget

Synopsis: Notify client controls when the action updates itself.

Declaration: `procedure UpdateTarget (Target: TObject); Virtual`

Visibility: `public`

Description: `UpdateTarget` should update the client control specified by `Target` when the action updates itself. In `TBasicAction`, the implementation of `UpdateTarget` is empty. Descendant classes should override and implement `UpdateTarget` to actually update the `Target` object.

An application programmer should never need to call `HandlesTarget` directly, it will be called by the action itself when needed.

See also: `HandlesTarget` (350), `ExecuteTarget` (351)

4.38.8 TBasicAction.ExecuteTarget

Synopsis: Executes the action on the `Target` object.

Declaration: `procedure ExecuteTarget (Target: TObject); Virtual`

Visibility: `public`

Description: `ExecuteTarget` performs the action on the `Target` object. In `TBasicAction` this method does nothing. Descendant classes should implement the action to be performed. For instance an action to post data in a dataset could call the `Post` method of the dataset.

An application programmer should never call `ExecuteTarget` directly.

See also: `HandlesTarget` (350), `UpdateTarget` (351), `Execute` (351)

4.38.9 TBasicAction.Execute

Synopsis: Triggers the `OnExecute` (353) event.

Declaration: `function Execute : Boolean; Dynamic`

Visibility: `public`

Description: `Execute` triggers the `OnExecute` event, if one is assigned. It returns `True` if the event handler was called, `False` otherwise.

4.38.10 TBasicAction.RegisterChanges

Synopsis: Registers a new client with the action.

Declaration: `procedure RegisterChanges (Value: TBasicActionLink)`

Visibility: `public`

Description: `RegisterChanges` adds `Value` to the list of clients.

See also: `UnregisterChanges` (352)

4.38.11 TBasicAction.UnRegisterChanges

Synopsis: Unregisters a client from the list of clients.

Declaration: `procedure UnRegisterChanges (Value: TBasicActionLink)`

Visibility: `public`

Description: `UnregisterChanges` removes `Value` from the list of clients. This is called for instance when the action is destroyed, or when the client is assigned a new action.

See also: `UnregisterChanges` ([352](#)), `Destroy` ([350](#))

4.38.12 TBasicAction.Update

Synopsis: Triggers the `OnUpdate` ([353](#)) event.

Declaration: `function Update : Boolean; Virtual`

Visibility: `public`

Description: `Update` triggers the `OnUpdate` event, if one is assigned. It returns `True` if the event was triggered, or `False` if no event was assigned.

Application programmers should never run `Update` directly. The `Update` method is called automatically by the action mechanism; Normally this is in the Idle time of an application. An application programmer should assign the `OnUpdate` ([353](#)) event, and perform any checks in that handler.

See also: `OnUpdate` ([353](#)), `Execute` ([351](#)), `UpdateTarget` ([351](#))

4.38.13 TBasicAction.Suspended

Declaration: `function Suspended : Boolean; Virtual`

Visibility: `public`

4.38.14 TBasicAction.ActionComponent

Synopsis: Returns the component that initiated the action.

Declaration: `Property ActionComponent : TComponent`

Visibility: `public`

Access: `Read,Write`

Description: `ActionComponent` is set to the component that caused the action to execute, e.g. a toolbar button or a menu item. The property is set just before the action executes, and is reset to nil after the action was executed.

See also: `Execute` ([351](#)), `OnExecute` ([353](#))

4.38.15 TBasicAction.OnExecute

Synopsis: Event triggered when the action executes.

Declaration: Property OnExecute : TNotifyEvent

Visibility: public

Access: Read,Write

Description: OnExecute is the event triggered when the action is activated (executed). The event is triggered e.g. when the user clicks e.g. on a menu item or a button associated to the action. The application programmer should provide a OnExecute event handler to execute whatever code is necessary when the button is pressed or the menu item is chosen.

Note that assigning an OnExecute handler will result in the Execute (351) method returning a True value. Predefined actions (such as dataset actions) will check the result of Execute and will not perform their normal task if the OnExecute handler was called.

See also: Execute (351), OnUpdate (353)

4.38.16 TBasicAction.OnUpdate

Synopsis: Event triggered when the application is idle.

Declaration: Property OnUpdate : TNotifyEvent

Visibility: public

Access: Read,Write

Description: OnUpdate is the event triggered when the application is idle, and the action is being updated. The OnUpdate event can be used to set the state of the action, for instance disable it if the action cannot be executed at this point in time.

See also: Update (352), OnExecute (353)

4.38.17 TBasicAction.Clients

Declaration: Property Clients[Idx: Integer]: TObject

Visibility: public

Access: Read

4.39 TBasicActionLink

4.39.1 Description

TBasicActionLink links an Action to its clients. With each client for an action, a TBasicActionLink class is instantiated to handle the communication between the action and the client. It passes events between the action and its clients, and thus presents the action with a uniform interface to the clients.

An application programmer should never use a TBasicActionLink instance directly; They are created automatically when an action is associated with a component. Component programmers should create specialized descendants of TBasicActionLink which communicate changes in the action to the component.

See also: TBasicAction (349)

4.39.2 Method overview

Page	Method	Description
354	Create	Creates a new instance of the TBasicActionLink class.
354	Destroy	Destroys the TBasicActionLink instance.
354	Execute	Calls the action's Execute method.
355	Update	Calls the action's Update method.

4.39.3 Property overview

Page	Properties	Access	Description
355	Action	rw	The action to which the link was assigned.
355	OnChange	rw	Event handler triggered when the action's properties change.

4.39.4 TBasicActionLink.Create

Synopsis: Creates a new instance of the TBasicActionLink class.

Declaration: `constructor Create(AClient: TObject); Virtual`

Visibility: `public`

Description: `Create` creates a new instance of a TBasicActionLink and assigns `AClient` as the client of the link.

Application programmers should never instantiate TBasicActionLink classes directly. An instance is created automatically when an action is assigned to a control (client).

Component programmers can override the create constructor to initialize further properties.

See also: `Destroy` ([354](#))

4.39.5 TBasicActionLink.Destroy

Synopsis: Destroys the TBasicActionLink instance.

Declaration: `destructor Destroy; Override`

Visibility: `public`

Description: `Destroy` unregisters the TBasicActionLink with the action, and then calls the inherited destructor.

Application programmers should never call `Destroy` directly. If a link should be destroyed at all, the `Free` method should be called instead.

See also: `Create` ([354](#))

4.39.6 TBasicActionLink.Execute

Synopsis: Calls the action's Execute method.

Declaration: `function Execute(AComponent: TComponent) : Boolean; Virtual`

Visibility: `public`

Description: `Execute` sets the `ActionComponent` (352) property of the associated `Action` (355) to `AComponent` and then calls the `Action`'s `execute` (351) method. After the action has executed, the `ActionComponent` property is cleared again.

The return value of the function is the return value of the `Action`'s `execute` method.

Application programmers should never call `Execute` directly. This method will be called automatically when the associated control is activated. (e.g. a button is clicked on)

Component programmers should call `Execute` whenever the action should be activated.

See also: `Action` (355), `TBasicAction.ActionComponent` (352), `TBasicAction.Execute` (351), `TBasicAction.onExecute` (353)

4.39.7 TBasicActionLink.Update

Synopsis: Calls the action's `Update` method.

Declaration: `function Update : Boolean; Virtual`

Visibility: `public`

Description: `Update` calls the associated `Action`'s `Update` (352) method.

Component programmers can override the `Update` method to provide additional processing when the `Update` method occurs.

4.39.8 TBasicActionLink.Action

Synopsis: The action to which the link was assigned.

Declaration: `Property Action : TBasicAction`

Visibility: `public`

Access: `Read,Write`

Description: `Action` represents the `Action` (349) which was assigned to the client. Setting this property will unregister the client at the old action (if one existed) and registers the client at the new action.

See also: `TBasicAction` (349)

4.39.9 TBasicActionLink.OnChange

Synopsis: Event handler triggered when the action's properties change.

Declaration: `Property OnChange : TNotifyEvent`

Visibility: `public`

Access: `Read,Write`

Description: `OnChange` is the event triggered when the action's properties change.

Application programmers should never need to assign this event. Component programmers can assign this event to have a client control reflect any changes in an `Action`'s properties.

See also: `Change` (353), `TBasicAction.Change` (349)

4.40 TBinaryObjectReader

4.40.1 Description

The `TBinaryObjectReader` class reads component data stored in binary form in a file. For this, it overrides or implements all abstract methods from `TAbstractObjectReader` (335). No new functionality is added by this class, it is a driver class for the streaming system.

It should never be necessary to create an instance of this class directly. Instead, the `TStream.WriteComponent` (471) call should be used.

See also: `TAbstractObjectReader` (335), `TBinaryObjectWriter` (362)

4.40.2 Method overview

Page	Method	Description
358	<code>BeginComponent</code>	Start reading a component.
358	<code>BeginProperty</code>	Start reading a property.
357	<code>BeginRootComponent</code>	Start reading the root component.
356	<code>Create</code>	Creates a new binary data reader instance.
357	<code>Destroy</code>	Destroys the binary data reader.
357	<code>NextValue</code>	Return the type of the next value.
358	<code>Read</code>	Read raw data from stream.
358	<code>ReadBinary</code>	Start reading a binary value.
359	<code>ReadCurrency</code>	Read a currency value from the stream.
359	<code>ReadDate</code>	Read a date.
359	<code>ReadDouble</code>	
358	<code>ReadFloat</code>	Read a float value.
359	<code>ReadIdent</code>	Read an identifier.
360	<code>ReadInt16</code>	Read a 16-bits integer.
360	<code>ReadInt32</code>	Read a 32-bits integer.
360	<code>ReadInt64</code>	Read a 64-bits integer.
360	<code>ReadInt8</code>	Read an 8-bits integer.
361	<code>ReadSet</code>	Read a set.
361	<code>ReadSignature</code>	Reads the filer signature.
359	<code>ReadSingle</code>	Read a single-size float value.
361	<code>ReadStr</code>	Read a short string.
361	<code>ReadString</code>	Read a string.
362	<code>ReadUnicodeString</code>	Read a Unicode string value.
357	<code>ReadValue</code>	Read the next value in the stream.
362	<code>ReadWideString</code>	Read a widestring value from the stream.
362	<code>SkipComponent</code>	Skip a component's data.
362	<code>SkipValue</code>	Skip a value's data.

4.40.3 TBinaryObjectReader.Create

Synopsis: Creates a new binary data reader instance.

Declaration: `constructor Create(Stream: TStream; BufSize: Integer)`

Visibility: `public`

Description: `Create` instantiates a new binary component data reader. The `Stream` stream is the stream from which data will be read. The `BufSize` argument is the size of the internal buffer that will be used by the reader. This can be used to optimize the reading process.

See also: [TAbstractObjectReader \(335\)](#)

4.40.4 TBinaryObjectReader.Destroy

Synopsis: Destroys the binary data reader.

Declaration: `destructor Destroy; Override`

Visibility: `public`

Description: `Destroy` frees the buffer allocated when the instance was created. It also positions the stream on the last used position in the stream (the buffering may cause the reader to read more bytes than were actually used.)

See also: [TBinaryObjectReader.Create \(356\)](#)

4.40.5 TBinaryObjectReader.NextValue

Synopsis: Return the type of the next value.

Declaration: `function NextValue : TValueType; Override`

Visibility: `public`

Description: `NextValue` returns the type of the next value in a binary stream, but does not read the value.

This method is simply the implementation for a binary stream of the abstract method introduced in [TAbstractObjectReader \(335\)](#)

4.40.6 TBinaryObjectReader.ReadValue

Synopsis: Read the next value in the stream.

Declaration: `function ReadValue : TValueType; Override`

Visibility: `public`

Description: `NextValue` reads the next value in a binary stream and returns the type of the read value.

This method is simply the implementation for a binary stream of the abstract method introduced in [TAbstractObjectReader \(335\)](#)

4.40.7 TBinaryObjectReader.BeginRootComponent

Synopsis: Start reading the root component.

Declaration: `procedure BeginRootComponent; Override`

Visibility: `public`

Description: `BeginRootComponent` starts reading the root component in a binary stream.

This method is simply the implementation for a binary stream of the abstract method introduced in [TAbstractObjectReader \(335\)](#)

4.40.8 TBinaryObjectReader.BeginComponent

Synopsis: Start reading a component.

Declaration: `procedure BeginComponent (var Flags: TFileFlags;
var AChildPos: Integer;
var CompClassName: string; var CompName: string)
; Override`

Visibility: public

Description: This method is simply the implementation for a binary stream of the abstract method introduced in TAbstractObjectReader ([335](#))

4.40.9 TBinaryObjectReader.BeginProperty

Synopsis: Start reading a property.

Declaration: `function BeginProperty : string; Override`

Visibility: public

Description: This method is simply the implementation for a binary stream of the abstract method introduced in TAbstractObjectReader ([335](#))

4.40.10 TBinaryObjectReader.Read

Synopsis: Read raw data from stream.

Declaration: `procedure Read (var Buf; Count: LongInt); Override`

Visibility: public

Description: Read is introduced for Delphi compatibility to read raw data from the component stream. This should not be used in production code as it will totally mess up the streaming.

See also: TAbstractObjectReader.Read ([338](#)), TReader.Read ([455](#))

4.40.11 TBinaryObjectReader.ReadBinary

Synopsis: Start reading a binary value.

Declaration: `procedure ReadBinary (const DestData: TMemoryStream); Override`

Visibility: public

Description: ReadBinary reads a binary value from a binary stream.

This method is simply the implementation for a binary stream of the abstract method introduced in TAbstractObjectReader ([335](#))

4.40.12 TBinaryObjectReader.ReadFloat

Synopsis: Read a float value.

Declaration: `function ReadFloat : Extended; Override`

Visibility: public

Description: `ReadFloat` reads a float value from a binary stream.

This method is simply the implementation for a binary stream of the abstract method introduced in `TAbstractObjectReader` ([335](#))

4.40.13 `TBinaryObjectReader.ReadSingle`

Synopsis: Read a single-size float value.

Declaration: `function ReadSingle : Single; Override`

Visibility: `public`

Description: `ReadSingle` reads a single-sized float value from a binary stream.

This method is simply the implementation for a binary stream of the abstract method introduced in `TAbstractObjectReader` ([335](#))

4.40.14 `TBinaryObjectReader.ReadDouble`

Declaration: `function ReadDouble : Double; Override`

Visibility: `public`

4.40.15 `TBinaryObjectReader.ReadDate`

Synopsis: Read a date.

Declaration: `function ReadDate : TDateTime; Override`

Visibility: `public`

Description: `ReadDate` reads a date value from a binary stream.

This method is simply the implementation for a binary stream of the abstract method introduced in `TAbstractObjectReader` ([335](#))

4.40.16 `TBinaryObjectReader.ReadCurrency`

Synopsis: Read a currency value from the stream.

Declaration: `function ReadCurrency : Currency; Override`

Visibility: `public`

Description: `var>ReadCurrency` reads a currency-typed value from a binary stream. It is the implementation of the method introduced in `TAbstractObjectReader` ([335](#)).

See also: `TAbstractObjectReader.ReadCurrency` ([339](#)), `TBinaryObjectWriter.WriteCurrency` ([366](#))

4.40.17 `TBinaryObjectReader.ReadIdent`

Synopsis: Read an identifier.

Declaration: `function ReadIdent (ValueType: TValueType) : string; Override`

Visibility: `public`

Description: `ReadIdent` reads an identifier from a binary stream.

This method is simply the implementation for a binary stream of the abstract method introduced in `TAbstractObjectReader` ([335](#))

4.40.18 TBinaryObjectReader.ReadInt8

Synopsis: Read an 8-bits integer.

Declaration: `function ReadInt8 : ShortInt; Override`

Visibility: `public`

Description: `Read8Int` reads an 8-bits signed integer from a binary stream.

This method is simply the implementation for a binary stream of the abstract method introduced in `TAbstractObjectReader` ([335](#))

4.40.19 TBinaryObjectReader.ReadInt16

Synopsis: Read a 16-bits integer.

Declaration: `function ReadInt16 : SmallInt; Override`

Visibility: `public`

Description: `Read16Int` reads a 16-bits signed integer from a binary stream.

This method is simply the implementation for a binary stream of the abstract method introduced in `TAbstractObjectReader` ([335](#))

4.40.20 TBinaryObjectReader.ReadInt32

Synopsis: Read a 32-bits integer.

Declaration: `function ReadInt32 : LongInt; Override`

Visibility: `public`

Description: `Read32Int` reads a 32-bits signed integer from a binary stream.

This method is simply the implementation for a binary stream of the abstract method introduced in `TAbstractObjectReader` ([335](#))

4.40.21 TBinaryObjectReader.ReadInt64

Synopsis: Read a 64-bits integer.

Declaration: `function ReadInt64 : Int64; Override`

Visibility: `public`

Description: `Read64Int` reads a 64-bits signed integer from a binary stream.

This method is simply the implementation for a binary stream of the abstract method introduced in `TAbstractObjectReader` ([335](#))

4.40.22 TBinaryObjectReader.ReadSet

Synopsis: Read a set.

Declaration: `function ReadSet (EnumType: Pointer) : Integer; Override`

Visibility: `public`

Description: `ReadSet` reads a set from a binary stream.

This method is simply the implementation for a binary stream of the abstract method introduced in `TAbstractObjectReader` (335)

4.40.23 TBinaryObjectReader.ReadSignature

Synopsis: Reads the filer signature.

Declaration: `procedure ReadSignature; Override`

Visibility: `public`

Description: `ReadSignature` is overridden by `TBinaryObjectReader` to read the signature (TPF0) from binary streams.

Errors: If the stream does not start with the correct signature, an `EReadError` (321) exception is raised.

See also: `TAbstractObjectReader.ReadSignature` (342), `TAbstractObjectWriter.WriteSignature` (345), `TBinaryObjectWriter.WriteSignature` (364)

4.40.24 TBinaryObjectReader.ReadStr

Synopsis: Read a short string.

Declaration: `function ReadStr : string; Override`

Visibility: `public`

Description: `ReadStr` reads a short string from a binary stream.

This method is simply the implementation for a binary stream of the abstract method introduced in `TAbstractObjectReader` (335)

4.40.25 TBinaryObjectReader.ReadString

Synopsis: Read a string.

Declaration: `function ReadString (StringType: TValueType) : string; Override`

Visibility: `public`

Description: `ReadStr` reads a string of type `StringType` from a binary stream.

This method is simply the implementation for a binary stream of the abstract method introduced in `TAbstractObjectReader` (335)

4.40.26 TBinaryObjectReader.ReadWideString

Synopsis: Read a widestring value from the stream.

Declaration: `function ReadWideString : WideString; Override`

Visibility: public

Description: `var>ReadWideString` reads a widestring-typed value from a binary stream. It is the implementation of the method introduced in `TAbstractObjectReader` (335).

See also: `TAbstractObjectReader.ReadWideString` (343), `TBinaryObjectWriter.WriteWideString` (367)

4.40.27 TBinaryObjectReader.ReadUnicodeString

Synopsis: Read a Unicode string value.

Declaration: `function ReadUnicodeString : UnicodeString; Override`

Visibility: public

Description: `ReadUnicodeString` is overridden by `TBinaryObjectReader` to read a `UnicodeString` value from the binary stream.

See also: `TAbstractObjectReader.ReadUnicodeString` (343)

4.40.28 TBinaryObjectReader.SkipComponent

Synopsis: Skip a component's data.

Declaration: `procedure SkipComponent (SkipComponentInfos: Boolean); Override`

Visibility: public

Description: `SkipComponent` skips the data of a component in a binary stream.

This method is simply the implementation for a binary stream of the abstract method introduced in `TAbstractObjectReader` (335).

4.40.29 TBinaryObjectReader.SkipValue

Synopsis: Skip a value's data.

Declaration: `procedure SkipValue; Override`

Visibility: public

Description: `SkipComponent` skips the data of the next value in a binary stream.

This method is simply the implementation for a binary stream of the abstract method introduced in `TAbstractObjectReader` (335)

4.41 TBinaryObjectWriter

4.41.1 Description

Driver class which stores component data in binary form.

4.41.2 Method overview

Page	Method	Description
364	BeginCollection	Start writing a collection.
364	BeginComponent	Start writing a component.
364	BeginList	Start writing a list.
365	BeginProperty	Start writing a property.
363	Create	Creates a new instance of a binary object writer.
363	Destroy	Destroys an instance of the binary object writer.
364	EndList	Mark the end of a list.
365	EndProperty	Marks the end of writing of a property.
364	FlushBuffer	Flush the buffer
365	Write	Write raw data to stream.
365	WriteBinary	Writes binary data to the stream.
365	WriteBoolean	Writes a boolean value to the stream.
366	WriteCurrency	Write a currency-valued type to a stream.
366	WriteDate	Writes a date type to the stream.
365	WriteFloat	Writes a float value to the stream.
366	WriteIdent	Writes an identifier to the stream.
366	WriteInteger	Writes an integer value to the stream.
367	WriteMethodName	Writes a methodname to the stream.
367	WriteSet	Writes a set value to the stream.
364	WriteSignature	Write stream signature to the stream.
366	WriteSingle	Writes a single-type real value to the stream.
367	WriteStr	Write a string to the binary stream.
367	WriteString	Writes a string value to the stream.
366	WriteUInt64	Write an unsigned 64-bit integer.
367	WriteUnicodeString	Write a Unicode string to the stream.
368	WriteVariant	Write a variant to the stream.
367	WriteWideString	Write a widestring-valued type to a stream.

4.41.3 Property overview

Page	Properties	Access	Description
368	Version	rw	

4.41.4 TBinaryObjectWriter.Create

Synopsis: Creates a new instance of a binary object writer.

Declaration: constructor `Create(Stream: TStream; BufSize: Integer)`

Visibility: public

Description: Creates a new instance of a binary object writer.

4.41.5 TBinaryObjectWriter.Destroy

Synopsis: Destroys an instance of the binary object writer.

Declaration: destructor `Destroy; Override`

Visibility: public

Description: Destroys an instance of the binary object writer.

4.41.6 TBinaryObjectWriter.WriteSignature

Synopsis: Write stream signature to the stream.

Declaration: `procedure WriteSignature; Override`

Visibility: `public`

Description: `WriteSignature` is overridden by `TBinaryObjectWriter` to write the signature (TPF0) to binary streams.

See also: `TAbstractObjectWriter.WriteSignature` (345), `TAbstractObjectReader.ReadSignature` (342), `TBinaryObjectReader.ReadSignature` (361)

4.41.7 TBinaryObjectWriter.FlushBuffer

Synopsis: Flush the buffer

Declaration: `procedure FlushBuffer; Override`

Visibility: `public`

Description: `FlushBuffer` flushes the buffer. It is provided for Delphi compatibility, and is not used in FPC.

See also: `TFile.FlushBuffer` (404), `TWriter.FlushBuffer` (538), `TAbstractObjectWriter.FlushBuffer` (346)

4.41.8 TBinaryObjectWriter.BeginCollection

Synopsis: Start writing a collection.

Declaration: `procedure BeginCollection; Override`

Visibility: `public`

4.41.9 TBinaryObjectWriter.BeginComponent

Synopsis: Start writing a component.

Declaration: `procedure BeginComponent (Component: TComponent; Flags: TFileFlags;
ChildPos: Integer); Override`

Visibility: `public`

4.41.10 TBinaryObjectWriter.BeginList

Synopsis: Start writing a list.

Declaration: `procedure BeginList; Override`

Visibility: `public`

4.41.11 TBinaryObjectWriter.EndList

Synopsis: Mark the end of a list.

Declaration: `procedure EndList; Override`

Visibility: `public`

4.41.12 TBinaryObjectWriter.BeginProperty

Synopsis: Start writing a property.

Declaration: `procedure BeginProperty(const PropName: string); Override`

Visibility: `public`

4.41.13 TBinaryObjectWriter.EndProperty

Synopsis: Marks the end of writing of a property.

Declaration: `procedure EndProperty; Override`

Visibility: `public`

4.41.14 TBinaryObjectWriter.Write

Synopsis: Write raw data to stream.

Declaration: `procedure Write(const Buffer; Count: LongInt); Override`

Visibility: `public`

Description: `Write` is introduced for Delphi compatibility to write raw data to the component stream. This should not be used in new production code as it will totally mess up the streaming.

See also: `TAbstractObjectWriter.Write` ([346](#)), `TWriter.Write` ([539](#))

4.41.15 TBinaryObjectWriter.WriteBinary

Synopsis: Writes binary data to the stream.

Declaration: `procedure WriteBinary(const Buffer; Count: LongInt); Override`

Visibility: `public`

4.41.16 TBinaryObjectWriter.WriteBoolean

Synopsis: Writes a boolean value to the stream.

Declaration: `procedure WriteBoolean(Value: Boolean); Override`

Visibility: `public`

4.41.17 TBinaryObjectWriter.WriteFloat

Synopsis: Writes a float value to the stream.

Declaration: `procedure WriteFloat(const Value: Extended); Override`

Visibility: `public`

4.41.18 TBinaryObjectWriter.WriteSingle

Synopsis: Writes a single-type real value to the stream.

Declaration: `procedure WriteSingle(const Value: Single); Override`

Visibility: public

4.41.19 TBinaryObjectWriter.WriteDate

Synopsis: Writes a date type to the stream.

Declaration: `procedure WriteDate(const Value: TDateTime); Override`

Visibility: public

4.41.20 TBinaryObjectWriter.WriteCurrency

Synopsis: Write a currency-valued type to a stream.

Declaration: `procedure WriteCurrency(const Value: Currency); Override`

Visibility: public

Description: `WriteCurrency` writes a currency-typed value to a binary stream. It is the implementation of the method introduced in `TAbstractObjectWriter` ([344](#)).

See also: `TAbstractObjectWriter.WriteCurrency` ([347](#))

4.41.21 TBinaryObjectWriter.WriteIdent

Synopsis: Writes an identifier to the stream.

Declaration: `procedure WriteIdent(const Ident: string); Override`

Visibility: public

4.41.22 TBinaryObjectWriter.WriteInteger

Synopsis: Writes an integer value to the stream.

Declaration: `procedure WriteInteger(Value: Int64); Override`

Visibility: public

4.41.23 TBinaryObjectWriter.WriteUInt64

Synopsis: Write an unsigned 64-bit integer.

Declaration: `procedure WriteUInt64(Value: QWord); Override`

Visibility: public

Description: `WriteUInt64` is overridden by `TBinaryObjectWriter` to write an unsigned 64-bit integer (QWord) to the stream. It tries to use the smallest possible storage for the value that is passed. (largest valuetype will be `vaQWord`).

See also: `TAbstractObjectWriter.WriteUInt64` ([348](#))

4.41.24 TBinaryObjectWriter.WriteMethodName

Synopsis: Writes a methodname to the stream.

Declaration: `procedure WriteMethodName(const Name: string); Override`

Visibility: public

4.41.25 TBinaryObjectWriter.WriteSet

Synopsis: Writes a set value to the stream.

Declaration: `procedure WriteSet(Value: LongInt; SetType: Pointer); Override`

Visibility: public

4.41.26 TBinaryObjectWriter.WriteString

Synopsis: Write a string to the binary stream.

Declaration: `procedure WriteStr(const Value: string)`

Visibility: public

Description: `WriteStr` writes a string value to the binary stream. It is exposed so it can be used in `DefineProperties`.

4.41.27 TBinaryObjectWriter.WriteString

Synopsis: Writes a string value to the stream.

Declaration: `procedure WriteString(const Value: string); Override`

Visibility: public

4.41.28 TBinaryObjectWriter.WriteWideString

Synopsis: Write a widestring-valued type to a stream.

Declaration: `procedure WriteWideString(const Value: WideString); Override`

Visibility: public

Description: `WriteWidestring` writes a widestring-typed value to a binary stream. It is the implementation of the method introduced in `TAbstractObjectWriter` (344).

See also: `TAbstractObjectWriter.WriteString` (349)

4.41.29 TBinaryObjectWriter.WriteUnicodeString

Synopsis: Write a Unicode string to the stream.

Declaration: `procedure WriteUnicodeString(const Value: UnicodeString); Override`

Visibility: public

Description: `WriteUnicodeString` is overridden `TBinaryObjectWriter` to write a `unicodestring` (valuetype `vaUString`) value to the stream. It simply writes the character length and then all widecharacters.

See also: `TAbstractObjectWriter.WriteString` (349)

4.41.30 TBinaryObjectWriter.WriteVariant

Synopsis: Write a variant to the stream.

Declaration: `procedure WriteVariant(const VarValue: Variant); Override`

Visibility: `public`

Description: `WriteVariant` is overridden by `TBinaryObjectWriter` to write a simple variant type to the stream. `WriteVariant` does not write arrays types or complex types. Only null, integer (ordinal) float and string types are written.

Errors: If a non-supported type is written, then an `EWriteError` exception is.

4.41.31 TBinaryObjectWriter.Version

Declaration: `Property Version : TBOWVersion`

Visibility: `public`

Access: `Read,Write`

4.42 TBits

4.42.1 Description

`TBits` can be used to store collections of bits in an indexed array. This is especially useful for storing collections of booleans: Normally the size of a boolean is the size of the smallest enumerated type, i.e. 1 byte. Since a bit can take 2 values it can be used to store a boolean as well. Since `TBits` can store 8 bits in a byte, it takes 8 times less space to store an array of booleans in a `TBits` class then it would take to store them in a conventional array.

`TBits` introduces methods to store and retrieve bit values, apply masks, and search for bits.

4.42.2 Method overview

Page	Method	Description
371	<code>AndBits</code>	Performs an <code>and</code> operation on the bits.
370	<code>Clear</code>	Clears a particular bit.
370	<code>Clearall</code>	Clears all bits in the array.
370	<code>CopyBits</code>	Copy bits from one bits set to another.
369	<code>Create</code>	Creates a new bits collection.
369	<code>Destroy</code>	Destroys a bit collection.
372	<code>Equals</code>	Determines whether the bits of 2 arrays are equal.
373	<code>FindFirstBit</code>	Find first bit with a particular value.
373	<code>FindNextBit</code>	Searches the next bit with a particular value.
374	<code>FindPrevBit</code>	Searches the previous bit with a particular value.
372	<code>Get</code>	Retrieve the value of a particular bit.
369	<code>GetFSize</code>	Returns the number of records used to store the bits.
372	<code>Grow</code>	Expands the bits array to the requested size.
372	<code>NotBits</code>	Performs a <code>not</code> operation on the bits.
374	<code>OpenBit</code>	Returns the position of the first bit that is set to <code>False</code> .
371	<code>OrBits</code>	Performs an <code>or</code> operation on the bits.
373	<code>SetIndex</code>	Sets the start position for <code>FindNextBit</code> (373) and <code>FindPrevBit</code> (374).
370	<code>SetOn</code>	Turn a particular bit on.
371	<code>XorBits</code>	Performs a <code>xor</code> operation on the bits.

4.42.3 Property overview

Page	Properties	Access	Description
374	Bits	rw	Access to all bits in the array.
375	Size	rw	Current size of the array of bits.

4.42.4 TBits.Create

Synopsis: Creates a new bits collection.

Declaration: `constructor Create(TheSize: LongInt); Virtual`

Visibility: `public`

Description: `Create` creates a new bit collection with initial size `TheSize`. The size of the collection can be changed later on.

All bits are initially set to zero.

See also: `Destroy` ([369](#))

4.42.5 TBits.Destroy

Synopsis: Destroys a bit collection.

Declaration: `destructor Destroy; Override`

Visibility: `public`

Description: `Destroy` destroys a previously created bit collection and releases all memory used to store the bit collection.

`Destroy` should never be called directly, `Free` should be used instead.

Errors: None.

See also: `Create` ([369](#))

4.42.6 TBits.GetFSize

Synopsis: Returns the number of records used to store the bits.

Declaration: `function GetFSize : SizeInt`

Visibility: `public`

Description: `GetFSize` returns the number of records used to store the current number of bits.

Errors: None.

See also: `Size` ([375](#))

4.42.7 TBits.SetOn

Synopsis: Turn a particular bit on.

Declaration: `procedure SetOn(Bit: SizeInt)`

Visibility: `public`

Description: `SetOn` turns on the bit at position `bit`, i.e. sets it to 1. If `bit` is at a position bigger than the current size, the collection is expanded to the required size using `Grow` (372).

Errors: If `bit` is larger than the maximum allowed bits array size or is negative, an `EBitsError` (319) exception is raised.

See also: `Bits` (374), `Clear` (370)

4.42.8 TBits.Clear

Synopsis: Clears a particular bit.

Declaration: `procedure Clear(Bit: SizeInt)`

Visibility: `public`

Description: `Clear` clears the bit at position `bit`. If the array `bit` is at a position bigger than the current size, the collection is expanded to the required size using `Grow` (372).

Errors: If `bit` is larger than the maximum allowed bits array size or is negative, an `EBitsError` (319) exception is raised.

See also: `Bits` (374), `seton` (370)

4.42.9 TBits.Clearall

Synopsis: Clears all bits in the array.

Declaration: `procedure Clearall`

Visibility: `public`

Description: `ClearAll` clears all bits in the array, i.e. sets them to zero. `ClearAll` works faster than clearing all individual bits, since it uses the packed nature of the bits.

Errors: None.

See also: `Bits` (374), `clear` (370)

4.42.10 TBits.CopyBits

Synopsis: Copy bits from one bits set to another.

Declaration: `procedure CopyBits(BitSet: TBits)`

Visibility: `public`

Description: `CopyBits` copies the bits from `BitSet` to the current `Bits`. Existing bits will be overwritten. The two sets of bits will be equal after this operation.

Errors: None.

See also: `TBits.Equals` (372)

4.42.11 TBits.AndBits

Synopsis: Performs an `and` operation on the bits.

Declaration: `procedure AndBits(BitSet: TBits)`

Visibility: `public`

Description: `andbits` performs an `and` operation on the bits in the array with the bits of array `BitSet`. If `BitSet` contains less bits than the current array, then all bits which have no counterpart in `BitSet` are cleared.

Errors: None.

See also: `ClearAll` (370), `OrBits` (371), `XOrBits` (371), `NotBits` (372)

4.42.12 TBits.OrBits

Synopsis: Performs an `or` operation on the bits.

Declaration: `procedure OrBits(BitSet: TBits)`

Visibility: `public`

Description: `andbits` performs an `or` operation on the bits in the array with the bits of array `BitSet`.

If `BitSet` contains less bits than the current array, then all bits which have no counterpart in `BitSet` are left untouched.

If the current array contains less bits than `BitSet` then it is grown to the size of `BitSet` before the `or` operation is performed.

Errors: None.

See also: `ClearAll` (370), `andBits` (371), `XOrBits` (371), `NotBits` (372)

4.42.13 TBits.XorBits

Synopsis: Performs a `xor` operation on the bits.

Declaration: `procedure XorBits(BitSet: TBits)`

Visibility: `public`

Description: `XorBits` performs a `xor` operation on the bits in the array with the bits of array `BitSet`.

If `BitSet` contains less bits than the current array, then all bits which have no counterpart in `BitSet` are left untouched.

If the current array contains less bits than `BitSet` then it is grown to the size of `BitSet` before the `xor` operation is performed.

Errors: None.

See also: `ClearAll` (370), `andBits` (371), `OrBits` (371), `NotBits` (372)

4.42.14 TBits.NotBits

Synopsis: Performs a not operation on the bits.

Declaration: `procedure NotBits(BitSet: TBits)`

Visibility: public

Description: `NotBits` performs a not operation on the bits in the array with the bits of array `BitSet`.

If `BitSet` contains less bits than the current array, then all bits which have no counterpart in `BitSet` are left untouched.

Errors: None.

See also: `ClearAll` (370), `andBits` (371), `OrBits` (371), `XOrBits` (371)

4.42.15 TBits.Get

Synopsis: Retrieve the value of a particular bit.

Declaration: `function Get(Bit: SizeInt) : Boolean`

Visibility: public

Description: `Get` returns `True` if the bit at position `bit` is set, or `False` if it is not set.

Errors: If `bit` is not a valid bit index then an `EBitsError` (319) exception is raised.

See also: `Bits` (374), `FindFirstBit` (373), `seton` (370)

4.42.16 TBits.Grow

Synopsis: Expands the bits array to the requested size.

Declaration: `procedure Grow(NBit: SizeInt)`

Visibility: public

Description: `Grow` expands the bit array so it can at least contain `nbit` bits. If `nbit` is less than the current size, nothing happens.

Errors: If there is not enough memory to complete the operation, then an `EBitsError` (319) is raised.

See also: `Size` (375)

4.42.17 TBits.Equals

Synopsis: Determines whether the bits of 2 arrays are equal.

Declaration: `function Equals(Obj: TObject) : Boolean; Override; Overload`
`function Equals(BitSet: TBits) : Boolean; Overload`

Visibility: public

Description: `equals` returns `True` if all the bits in `BitSet` are the same as the ones in the current `BitSet`; if not, `False` is returned.

If the sizes of the two `BitSets` are different, the arrays are still reported equal when all the bits in the larger set, which are not present in the smaller set, are zero.

Errors: None.

See also: `ClearAll` (370), `andBits` (371), `OrBits` (371), `XOrBits` (371)

4.42.18 TBits.SetIndex

Synopsis: Sets the start position for `FindNextBit` (373) and `FindPrevBit` (374).

Declaration: `procedure SetIndex(Index: SizeInt)`

Visibility: `public`

Description: `SetIndex` sets the search start position for `FindNextBit` (373) and `FindPrevBit` (374) to `Index`. This means that these calls will start searching from position `Index`.

This mechanism provides an alternative to `FindFirstBit` (373) which can also be used to position for the `FindNextBit` and `FindPrevBit` calls.

Errors: None.

See also: `FindNextBit` (373), `FindPrevBit` (374), `FindFirstBit` (373), `OpenBit` (374)

4.42.19 TBits.FindFirstBit

Synopsis: Find first bit with a particular value.

Declaration: `function FindFirstBit(State: Boolean) : SizeInt`

Visibility: `public`

Description: `FindFirstBit` searches for the first bit with value `State`. It returns the position of this bit, or `-1` if no such bit was found.

The search starts at position 0 in the array. If the first search returned a positive result, the found position is saved, and the `FindNextBit` (373) and `FindPrevBit` (374) will use this position to resume the search. To start a search from a certain position, the start position can be set with the `SetIndex` (373) instead.

Errors: None.

See also: `FindNextBit` (373), `FindPrevBit` (374), `OpenBit` (374), `SetIndex` (373)

4.42.20 TBits.FindNextBit

Synopsis: Searches the next bit with a particular value.

Declaration: `function FindNextBit : SizeInt`

Visibility: `public`

Description: `FindNextBit` resumes a previously started search. It searches for the next bit with the value specified in the `FindFirstBit` (373). The search is done towards the end of the array and starts at the position last reported by one of the `Find` calls or at the position set with `SetIndex` (373).

If another bit with the same value is found, its position is returned. If no more bits with the same value are present in the array, `-1` is returned.

Errors: None.

See also: `FindFirstBit` (373), `FindPrevBit` (374), `OpenBit` (374), `SetIndex` (373)

4.42.21 TBits.FindPrevBit

Synopsis: Searches the previous bit with a particular value.

Declaration: `function FindPrevBit : SizeInt`

Visibility: `public`

Description: `FindPrevBit` resumes a previously started search. It searches for the previous bit with the value specified in the `FindFirstBit` (373). The search is done towards the beginning of the array and starts at the position last reported by one of the `Find` calls or at the position set with `SetIndex` (373).

If another bit with the same value is found, its position is returned. If no more bits with the same value are present in the array, `-1` is returned.

Errors: None.

See also: `FindFirstBit` (373), `FindNextBit` (373), `OpenBit` (374), `SetIndex` (373)

4.42.22 TBits.OpenBit

Synopsis: Returns the position of the first bit that is set to `False`.

Declaration: `function OpenBit : SizeInt`

Visibility: `public`

Description: `OpenBit` returns the position of the first bit whose value is `0` (`False`), or `-1` if no open bit was found. This call is equivalent to `FindFirstBit(False)`, except that it doesn't set the position for the next searches.

Errors: None.

See also: `FindNextBit` (373), `FindPrevBit` (374), `FindFirstBit` (373), `SetIndex` (373)

4.42.23 TBits.Bits

Synopsis: Access to all bits in the array.

Declaration: `Property Bits[Bit: SizeInt]: Boolean; default`

Visibility: `public`

Access: `Read, Write`

Description: `Bits` allows indexed access to all of the bits in the array. It gives `True` if the bit is `1`, `False` otherwise; Assigning to this property will set, respectively clear the bit.

Errors: If an index is specified which is out of the allowed range then an `EBitsError` (319) exception is raised.

See also: `Size` (375)

4.42.24 TBits.Size

Synopsis: Current size of the array of bits.

Declaration: `Property Size : SizeInt`

Visibility: `public`

Access: `Read, Write`

Description: `Size` is the current size of the bit array. Setting this property will adjust the size; this is equivalent to calling `Grow(Value-1)`

Errors: If an invalid size (negative or too large) is specified, a `EBitsError` (319) exception is raised.

See also: `Bits` (374)

4.43 TBytesStream

4.43.1 Description

`TBytesStream` is a stream that uses an array of byte (`TBytes` (281)) to keep the stream data. it overrides the `TMemoryStream` (436) memory allocation routine to use the array of bytes. The array of bytes is exposed through the `Bytes` (281) property.

See also: `TBytes` (281), `TMemoryStream` (436), `Bytes` (281)

4.43.2 Method overview

Page	Method	Description
375	<code>Create</code>	Create a new instance of the stream, initializing it with an array of bytes.

4.43.3 Property overview

Page	Properties	Access	Description
375	<code>Bytes</code>	<code>r</code>	The stream data as an array of bytes.

4.43.4 TBytesStream.Create

Synopsis: Create a new instance of the stream, initializing it with an array of bytes.

Declaration: `constructor Create(const ABytes: TBytes); Virtual; Overload`

Visibility: `public`

Description: `Create` creates a new instance and initializes the memory with the data in `ABytes`.

See also: `TBytes` (281), `TMemoryStream` (436), `Bytes` (281)

4.43.5 TBytesStream.Bytes

Synopsis: The stream data as an array of bytes.

Declaration: `Property Bytes : TBytes`

Visibility: `public`

Access: Read

Description: `Bytes` provides byte-sized access to the array of bytes that represent the stream data. As a pointer value, it equals `TCustomMemoryStream.Memory` (400), meaning that `Memory` points to the first byte in the array.

See also: `TBytes` (281), `TMemoryStream` (436), `TCustomMemoryStream.Memory` (400)

4.44 TCollection

4.44.1 Description

`TCollection` implements functionality to manage a collection of named objects. Each of these objects needs to be a descendant of the `TCollectionItem` (383) class. Exactly which type of object is managed can be seen from the `TCollection.ItemClass` (381) property.

Normally, no `TCollection` is created directly. Instead, a descendants of `TCollection` and `TCollectionItem` (383) are created as a pair.

See also: `TCollectionItem` (383)

4.44.2 Method overview

Page	Method	Description
377	<code>Add</code>	Creates and adds a new item to the collection.
377	<code>Assign</code>	Assigns one collection to another.
378	<code>BeginUpdate</code>	Start an update batch.
378	<code>Clear</code>	Removes all items from the collection.
376	<code>Create</code>	Creates a new collection.
379	<code>Delete</code>	Delete an item from the collection.
377	<code>Destroy</code>	Destroys the collection and frees all the objects it manages.
379	<code>EndUpdate</code>	Ends an update batch.
380	<code>Exchange</code>	Exchange 2 items in the collection.
380	<code>FindItemID</code>	Searches for an Item in the collection, based on its <code>TCollectionItem.ID</code> (385) property.
379	<code>GetEnumerator</code>	Create an <code>IEnumerator</code> instance.
379	<code>GetNamePath</code>	Overrides <code>TPersistent.GetNamePath</code> (448) to return a proper pathname.
380	<code>Insert</code>	Insert an item in the collection.
380	<code>Move</code>	Move an item in the collection.
377	<code>Owner</code>	Owner of the collection.
381	<code>Sort</code>	Sort the items in the collection.

4.44.3 Property overview

Page	Properties	Access	Description
381	<code>Count</code>	r	Number of items in the collection.
381	<code>ItemClass</code>	r	Class pointer for each item in the collection.
381	<code>Items</code>	rw	Indexed array of items in the collection.

4.44.4 TCollection.Create

Synopsis: Creates a new collection.

Declaration: `constructor Create(AItemClass: TCollectionItemClass)`

Visibility: `public`

Description: `Create` instantiates a new instance of the `TCollection` class which will manage objects of class `AItemClass`. It creates the list used to hold all objects, and stores the `AItemClass` for the adding of new objects to the collection.

See also: `TCollection.ItemClass` (381), `TCollection.Destroy` (377)

4.44.5 TCollection.Destroy

Synopsis: Destroys the collection and frees all the objects it manages.

Declaration: `destructor Destroy; Override`

Visibility: `public`

Description: `Destroy` first clears the collection, and then frees all memory allocated to this instance. Don't call `Destroy` directly, call `Free` instead.

See also: `TCollection.Create` (376)

4.44.6 TCollection.Owner

Synopsis: Owner of the collection.

Declaration: `function Owner : TPersistent`

Visibility: `public`

Description: `Owner` returns a reference to the owner of the collection. This property is required by the object inspector to be able to show the collection.

4.44.7 TCollection.Add

Synopsis: Creates and adds a new item to the collection.

Declaration: `function Add : TCollectionItem`

Visibility: `public`

Description: `Add` instantiates a new item of class `TCollection.ItemClass` (381) and adds it to the list. The newly created object is returned.

See also: `TCollection.ItemClass` (381), `TCollection.Clear` (378)

4.44.8 TCollection.Assign

Synopsis: Assigns one collection to another.

Declaration: `procedure Assign(Source: TPersistent); Override`

Visibility: `public`

Description: `Assign` assigns the contents of one collection to another. It does this by clearing the items list, and adding as much elements as there are in the `Source` collection; it assigns to each created element the contents of it's counterpart in the `Source` element.

Two collections cannot be assigned to each other if instances of the `ItemClass` classes cannot be assigned to each other.

Errors: If the objects in the collections cannot be assigned to one another, then an `EConvertError` is raised.

See also: `TPersistent.Assign` (447), `TCollectionItem` (383)

4.44.9 TCollection.BeginUpdate

Synopsis: Start an update batch.

Declaration: `procedure BeginUpdate; Virtual`

Visibility: `public`

Description: `BeginUpdate` is called at the beginning of a batch update. It raises the update count with 1.

Call `BeginUpdate` at the beginning of a series of operations that will change the state of the collection. This will avoid the call to `TCollection.Update` (376) for each operation. At the end of the operations, a corresponding call to `EndUpdate` must be made. It is best to do this in the context of a `Try ... finally` block:

```
With MyCollection Do
  try
    BeginUpdate;
    // Some Lengthy operations
  finally
    EndUpdate;
  end;
```

This insures that the number of calls to `BeginUpdate` always matches the number of calls to `TCollection.EndUpdate` (379), even in case of an exception.

See also: `TCollection.EndUpdate` (379), `TCollection.Changed` (376), `TCollection.Update` (376)

4.44.10 TCollection.Clear

Synopsis: Removes all items from the collection.

Declaration: `procedure Clear`

Visibility: `public`

Description: `Clear` will clear the collection, i.e. each item in the collection is destroyed and removed from memory. After a call to `Clear`, `Count` is zero.

See also: `TCollection.Add` (377), `TCollectionItem.Destroy` (384), `TCollection.Destroy` (377)

4.44.11 TCollection.EndUpdate

Synopsis: Ends an update batch.

Declaration: `procedure EndUpdate; Virtual`

Visibility: `public`

Description: `EndUpdate` signals the end of a series of operations that change the state of the collection, possibly triggering an update event. It does this by decreasing the update count with 1 and calling `TCollection.Changed` (376) it should always be used in conjunction with `TCollection.BeginUpdate` (378), preferably in the `Finally` section of a `Try ... Finally` block.

See also: `TCollection.BeginUpdate` (378), `TCollection.Changed` (376), `TCollection.Update` (376)

4.44.12 TCollection.Delete

Synopsis: Delete an item from the collection.

Declaration: `procedure Delete(Index: Integer)`

Visibility: `public`

Description: `Delete` deletes the item at (zero based) position `Index` from the collection. This will result in a `cnDeleted` notification.

Errors: If an invalid index is specified, an `EListError` exception is raised.

See also: `TCollection.Items` (381), `TCollection.Insert` (380), `TCollection.Clear` (378)

4.44.13 TCollection.GetEnumerator

Synopsis: Create an `IEnumerator` instance.

Declaration: `function GetEnumerator : TCollectionEnumerator`

Visibility: `public`

Description: `GetEnumerator` is the implementation of the `IEnumerable` (1591) interface for `TCollection`. It creates a `TCollectionEnumerator` (382) instance and returns its `IEnumerator` (1591) interface.

See also: `TCollectionEnumerator` (382), `IEnumerator` (1591), `IEnumerable` (1591)

4.44.14 TCollection.GetNamePath

Synopsis: Overrides `TPersistent.GetNamePath` (448) to return a proper pathname.

Declaration: `function GetNamePath : string; Override`

Visibility: `public`

Description: `GetNamePath` returns the name path for this collection. If the following conditions are satisfied:

1. There is an owner object.
2. The owner object returns a non-empty name path.
3. The `TCollection.Propname` (376) property is not empty

collection has an owner and the owning object has a name, then the function returns the owner name, followed by the `propname`. If one of the conditions is not satisfied, then the classname is returned.

See also: `TCollection.GetOwner` (376), `TCollection.Propname` (376)

4.44.15 TCollection.Insert

Synopsis: Insert an item in the collection.

Declaration: `function Insert(Index: Integer) : TCollectionItem`

Visibility: public

Description: `Insert` creates a new item instance and inserts it in the collection at position `Index`, and returns the new instance.

In contrast, `TCollection.Add` (377) adds a new item at the end.

Errors: None.

See also: `TCollection.Add` (377), `TCollection.Delete` (379), `TCollection.Items` (381)

4.44.16 TCollection.FindItemID

Synopsis: Searches for an Item in the collection, based on its `TCollectionItem.ID` (385) property.

Declaration: `function FindItemID(ID: Integer) : TCollectionItem`

Visibility: public

Description: `FindItemID` searches through the collection for the item that has a value of `ID` for its `TCollectionItem.ID` (385) property, and returns the found item. If no such item is found in the collection, `Nil` is returned.

The routine performs a linear search, so this can be slow on very large collections.

See also: `TCollection.Items` (381), `TCollectionItem.ID` (385)

4.44.17 TCollection.Exchange

Synopsis: Exchange 2 items in the collection.

Declaration: `procedure Exchange(const Index1: Integer; const index2: Integer)`

Visibility: public

Description: `Exchange` exchanges the items at indexes `Index1` and `Index2` in the collection.

Errors: If one of the two indexes is invalid (less than zero or larger than the number of items) an `EListError` exception is raised.

See also: `Items` (381), `TCollectionItem.Index` (385)

4.44.18 TCollection.Move

Synopsis: Move an item in the collection.

Declaration: `procedure Move(const Index1: Integer; const index2: Integer)`

Visibility: public

Description: `Move` will move the collection item from position `index1` to `Index2`. In difference with `Exchange` (380) the indexes of the elements between `index1` to `Index2` will shift.

Errors: If an invalid index is specified in `index1` or `Index2`, an exception will be raised.

See also: `TCollection.Exchange` (380)

4.44.19 TCollection.Sort

Synopsis: Sort the items in the collection.

Declaration: `procedure Sort (const Compare: TCollectionSortCompare)`

Visibility: public

Description: `Sort` sorts the items in the collection, and uses the `Compare` procedure to compare 2 items in the collection. It is more efficient to use this method than to perform the sort manually, because the list items are manipulated directly.

For more information on how the `Compare` function should behave, see the `TCollectionSortCompare` (286) type.

See also: `TCollectionSortCompare` (286)

4.44.20 TCollection.Count

Synopsis: Number of items in the collection.

Declaration: `Property Count : Integer`

Visibility: public

Access: Read

Description: `Count` contains the number of items in the collection.

Remark The items in the collection are identified by their `TCollectionItem.Index` (385) property, which is a zero-based index, meaning that it can take values between 0 and `Count-1`, borders included.

See also: `TCollectionItem.Index` (385), `TCollection.Items` (381)

4.44.21 TCollection.ItemClass

Synopsis: Class pointer for each item in the collection.

Declaration: `Property ItemClass : TCollectionItemClass`

Visibility: public

Access: Read

Description: `ItemClass` is the class pointer with which each new item in the collection is created. It is the value that was passed to the collection's constructor when it was created, and does not change during the lifetime of the collection.

See also: `TCollectionItem` (383), `TCollection.Items` (381)

4.44.22 TCollection.Items

Synopsis: Indexed array of items in the collection.

Declaration: `Property Items[Index: Integer]: TCollectionItem`

Visibility: public

Access: Read,Write

Description: `Items` provides indexed access to the items in the collection. Since the array is zero-based, `Index` should be an integer between 0 and `Count-1`.

It is possible to set or retrieve an element in the array. When setting an element of the array, the object that is assigned should be compatible with the class of the objects in the collection, as given by the `TCollection.ItemClass` (381) property.

Adding an element to the array can be done with the `TCollection.Add` (377) method. The array can be cleared with the `TCollection.Clear` (378) method. Removing an element of the array should be done by freeing that element.

See also: `TCollection.Count` (381), `TCollection.ItemClass` (381), `TCollection.Clear` (378), `TCollection.Add` (377)

4.45 TCollectionEnumerator

4.45.1 Description

`TCollectionEnumerator` implements the `#rtl.system.IEnumerator` (1591) interface for the `TCollection` (376) class, so the `TCollection` class can be used in a `for ... in` loop. It is returned by the `TCollection.GetEnumerator` (379) method of `TCollection`.

See also: `TCollection` (376), `TCollection.GetEnumerator` (379), `#rtl.system.IEnumerator` (1591)

4.45.2 Method overview

Page	Method	Description
382	<code>Create</code>	Initialize a new instance of <code>TCollectionEnumerator</code> .
382	<code>GetCurrent</code>	Return the current pointer in the list.
383	<code>MoveNext</code>	Move the position of the enumerator to the next position in the collection.

4.45.3 Property overview

Page	Properties	Access	Description
383	<code>Current</code>	<code>r</code>	Current pointer in the list.

4.45.4 TCollectionEnumerator.Create

Synopsis: Initialize a new instance of `TCollectionEnumerator`.

Declaration: `constructor Create (ACollection: TCollection)`

Visibility: `public`

Description: `Create` initializes a new instance of `TCollectionEnumerator` and keeps a reference to the collection `ACollection` that will be enumerated.

See also: `TCollection` (376)

4.45.5 TCollectionEnumerator.GetCurrent

Synopsis: Return the current pointer in the list.

Declaration: `function GetCurrent : TCollectionItem`

Visibility: public

Description: `GetCurrent` returns the current `TCollectionItem` (383) instance in the enumerator.

Errors: No checking is done on the validity of the current position.

See also: `MoveNext` (383), `TCollectionItem` (383)

4.45.6 `TCollectionEnumerator.MoveNext`

Synopsis: Move the position of the enumerator to the next position in the collection.

Declaration: `function MoveNext : Boolean`

Visibility: public

Description: `MoveNext` puts the pointer on the next item in the collection, and returns `True` if this succeeded, or `False` if the pointer is past the last element in the list.

Errors: Note that if `False` is returned, calling `GetCurrent` will result in an exception.

See also: `GetCurrent` (382)

4.45.7 `TCollectionEnumerator.Current`

Synopsis: Current pointer in the list.

Declaration: `Property Current : TCollectionItem`

Visibility: public

Access: Read

Description: `Current` redefines `GetCurrent` (382) as a property.

See also: `GetCurrent` (382)

4.46 `TCollectionItem`

4.46.1 Description

`TCollectionItem` and `TCollection` (376) form a pair of base classes that manage a collection of named objects. The `TCollectionItem` is the named object that is managed, it represents one item in the collection. An item in the collection is represented by three properties: `TCollectionItem.DisplayName` (386), `TCollection.Index` (376) and `TCollectionItem.ID` (385).

A `TCollectionItem` object is never created directly. To manage a set of named items, it is necessary to make a descendant of `TCollectionItem` to which needed properties and methods are added. This descendant can then be managed with a `TCollection` (376) class. The managing collection will create and destroy its items by itself, it should therefore never be necessary to create `TCollectionItem` descendants manually.

See also: `TCollection` (376)

4.46.2 Method overview

Page	Method	Description
384	Create	Creates a new instance of this collection item.
384	Destroy	Destroys this collection item.
384	GetNamePath	Returns the namepath of this collection item.

4.46.3 Property overview

Page	Properties	Access	Description
385	Collection	rw	Pointer to the collection managing this item.
386	DisplayName	rw	Name of the item, displayed in the object inspector.
385	ID	r	Initial index of this item.
385	Index	rw	Index of the item in its managing collection TCollection.Items (381) property.

4.46.4 TCollectionItem.Create

Synopsis: Creates a new instance of this collection item.

Declaration: `constructor Create(ACollection: TCollection); Virtual`

Visibility: `public`

Description: `Create` instantiates a new item in a `TCollection` ([376](#)). It is called by the `TCollection.Add` ([377](#)) function and should under normal circumstances never be called directly. called

See also: `TCollectionItem.Destroy` ([384](#))

4.46.5 TCollectionItem.Destroy

Synopsis: Destroys this collection item.

Declaration: `destructor Destroy; Override`

Visibility: `public`

Description: `Destroy` removes the item from the managing collection and Destroys the item instance. This is the only way to remove items from a collection;

See also: `TCollectionItem.Create` ([384](#))

4.46.6 TCollectionItem.GetNamePath

Synopsis: Returns the namepath of this collection item.

Declaration: `function GetNamePath : string; Override`

Visibility: `public`

Description: `GetNamePath` overrides the `TPersistent.GetNamePath` ([448](#)) method to return the name of the managing collection and appends its `TCollectionItem.Index` ([385](#)) property.

See also: `TCollectionItem.Collection` ([385](#)), `TPersistent.GetNamePath` ([448](#)), `TCollectionItem.Index` ([385](#))

4.46.7 TCollectionItem.Collection

Synopsis: Pointer to the collection managing this item.

Declaration: `Property Collection : TCollection`

Visibility: `public`

Access: `Read,Write`

Description: `Collection` points to the collection managing this item. This property can be set to point to a new collection. If this is done, the old collection will be notified that the item should no longer be managed, and the new collection is notified that it should manage this item as well.

See also: `TCollection` ([376](#))

4.46.8 TCollectionItem.ID

Synopsis: Initial index of this item.

Declaration: `Property ID : Integer`

Visibility: `public`

Access: `Read`

Description: `ID` is the initial value of `TCollectionItem.Index` ([385](#)); it doesn't change after the index changes. It can be used to uniquely identify the item. The `ID` property doesn't change as items are added and removed from the collection.

While the `TCollectionItem.Index` ([385](#)) property forms a continuous series, `ID` does not. If items are removed from the collection, their `ID` is not used again, leaving gaps. Only when the collection is initially created, the `ID` and `Index` properties will be equal.

See also: `TCollection.Items` ([381](#)), `TCollectionItem.Index` ([385](#))

4.46.9 TCollectionItem.Index

Synopsis: Index of the item in its managing collection `TCollection.Items` ([381](#)) property.

Declaration: `Property Index : Integer`

Visibility: `public`

Access: `Read,Write`

Description: `Index` is the current index of the item in its managing collection's `TCollection.Items` ([381](#)) property. This property may change as items are added and removed from the collection.

The index of an item is zero-based, i.e. the first item has index zero. The last item has index `Count-1` where `Count` is the number of items in the collection.

The `Index` property of the items in a collection form a continuous series ranging from 0 to `Count-1`. The `TCollectionItem.ID` ([385](#)) property does not form a continuous series, but can also be used to identify an item.

See also: `TCollectionItem.ID` ([385](#)), `TCollection.Items` ([381](#))

4.46.10 TCollectionItem.DisplayName

Synopsis: Name of the item, displayed in the object inspector.

Declaration: `Property DisplayName : string`

Visibility: `public`

Access: `Read,Write`

Description: `DisplayName` contains the name of this item as shown in the object inspector. For `TCollectionItem` this returns always the class name of the managing collection, followed by the index of the item.

`TCollectionItem` does not implement any functionality to store the `DisplayName` property. The property can be set, but this will have no effect other than that the managing collection is notified of a change. The actual displayname will remain unchanged. To store the `DisplayName` property, `TCollectionItem` descendants should override the `TCollectionItem.SetDisplayName` (383) and `TCollectionItem.GetDisplayName` (383) to add storage functionality.

See also: `TCollectionItem.Index` (385), `TCollectionItem.ID` (385), `TCollectionItem.GetDisplayName` (383), `TCollectionItem.SetDisplayName` (383)

4.47 TComponent

4.47.1 Description

`TComponent` is the base class for any set of classes that needs owner-owned functionality, and which needs support for property streaming. All classes that should be handled by an IDE (Integrated Development Environment) must descend from `TComponent`, as it includes all support for streaming all its published properties.

Components can 'own' other components. `TComponent` introduces methods for enumerating the child components. It also allows to name the owned components with a unique name. Furthermore, functionality for sending notifications when a component is removed from the list or removed from memory altogether is also introduced in `TComponent`.

`TComponent` introduces a form of automatic memory management: When a component is destroyed, all its child components will be destroyed first.

4.47.2 Interfaces overview

Page	Interfaces	Description
325	<code>IInterfaceComponentReference</code>	Interface for checking component references.
1593	<code>IUnknown</code>	Basic interface for all COM-based interfaces.

4.47.3 Method overview

Page	Method	Description
389	BeforeDestruction	Overrides standard BeforeDestruction.
388	Create	Creates a new instance of the component.
388	Destroy	Destroys the instance of the component.
389	DestroyComponents	Destroy child components.
389	Destroying	Called when the component is being destroyed.
389	ExecuteAction	Standard action execution method.
390	FindComponent	Finds and returns the named component in the owned components.
390	FreeNotification	Ask the component to notify called when it is being destroyed.
390	FreeOnRelease	Part of the <code>IVCLComObject</code> interface.
390	GetEnumerator	Create an <code>IEnumerator</code> instance.
391	GetNamePath	Returns the name path of this component.
391	GetParentComponent	Returns the parent component.
391	HasParent	Does the component have a parent ?
391	InsertComponent	Insert the given component in the list of owned components.
393	IsImplementorOf	Checks if the current component is the implementor of the interface.
387	Notification	Called by components that are freed and which received a <code>FreeNotification</code> .
393	ReferenceInterface	Interface implementation of Notification.
392	RemoveComponent	Remove the given component from the list of owned components.
390	RemoveFreeNotification	Remove a component from the Free Notification list.
392	SafeCallException	Part of the <code>IVCLComObject</code> Interface.
392	SetSubComponent	Sets the <code>csSubComponent</code> style.
392	UpdateAction	Updates the state of an action.
388	WriteState	Writes the component to a stream.

4.47.4 Property overview

Page	Properties	Access	Description
393	ComObject	r	Interface reference implemented by the component.
394	ComponentCount	r	Count of owned components.
394	ComponentIndex	rw	Index of component in it's owner's list.
393	Components	r	Indexed list (zero-based) of all owned components.
394	ComponentState	r	Current component's state.
395	ComponentStyle	r	Current component's style.
395	DesignInfo	rw	Information for IDE designer.
396	Name	rws	Name of the component.
395	Owner	r	Owner of this component.
396	Tag	rw	Tag value of the component.
395	VCLComObject	rw	Not implemented.

4.47.5 TComponent.Notification

Synopsis: Called by components that are freed and which received a `FreeNotification`.

Declaration: `procedure Notification(AComponent: TComponent; Operation: TOperation)`
`; Virtual`

Visibility: protected

Description: `Notification` is called whenever a child component is destroyed, inserted or removed from the list of owned component. Components that were requested to send a notification when they are freed ((with `FreeNotification` (390)) will also call `Notification` when they are freed.

The `AComponent` parameter specifies which component sends the notification, and `Operation` specifies whether the component is being inserted into or removed from the child component list, or whether it is being destroyed.

Descendants of `TComponent` can use `FreeNotification` (390) to request notification of the destruction of another object. By overriding the `Notification` method, they can do special processing (typically, set a reference to this component to `Nil`) when this component is destroyed. The `Notification` method is called quite often in the streaming process, so speed should be a consideration when overriding this method.

See also: `TOperation` (293), `TComponent.FreeNotification` (390), `TComponent.RemoveFreeNotification` (390)

4.47.6 TComponent.WriteState

Synopsis: Writes the component to a stream.

Declaration: `procedure WriteState(Writer: TWriter); Virtual`

Visibility: `public`

Description: `WriteState` writes the component's current state to a stream through the writer (537) object `writer`. Values for all published properties of the component can be written to the stream. Normally there is no need to call `WriteState` directly. The streaming system calls `WriteState` itself.

The `TComponent` (386) implementation of `WriteState` simply calls `TWriter.WriteData` (537). Descendant classes can, however, override `WriteState` to provide additional processing of stream data.

See also: `ReadState` (386), `TStream.WriteComponent` (471), `TWriter.WriteData` (537)

4.47.7 TComponent.Create

Synopsis: Creates a new instance of the component.

Declaration: `constructor Create(AOwner: TComponent); Virtual`

Visibility: `public`

Description: `Create` creates a new instance of a `TComponent` class. If `AOwner` is not `Nil`, the new component attempts to insert itself in the list of owned components of the owner.

See also: `Insert` (386), `Owner` (395)

4.47.8 TComponent.Destroy

Synopsis: Destroys the instance of the component.

Declaration: `destructor Destroy; Override`

Visibility: `public`

Description: `Destroy` sends a `opRemove` notification to all components in the free-notification list. After that, all owned components are destroyed by calling `DestroyComponents` (389) (and hence removed from the list of owned components). When this is done, the component removes itself from its owner's child component list. After that, the parent's destroy method is called.

See also: `Notification` (387), `Owner` (395), `DestroyComponents` (389), `Components` (393)

4.47.9 TComponent.BeforeDestruction

Synopsis: Overrides standard BeforeDestruction.

Declaration: `procedure BeforeDestruction; Override`

Visibility: `public`

Description: BeforeDestruction is overridden by TComponent to set the `csDestroying` flag in TComponent.ComponentState (394)

See also: TComponent.ComponentState (394)

4.47.10 TComponent.DestroyComponents

Synopsis: Destroy child components.

Declaration: `procedure DestroyComponents`

Visibility: `public`

Description: DestroyComponents calls the destructor of all owned components, till no more components are left in the Components (393) array.

Calling the destructor of an owned component has as the effect that the component will remove itself from the list of owned components, if nothing has disrupted the sequence of destructors.

Errors: If an overridden 'destroy' method does not call it's inherited destructor or raises an exception, it's TComponent.Destroy (388) destructor will not be called, which may result in an endless loop.

See also: Destroy (388), Components (393)

4.47.11 TComponent.Destroying

Synopsis: Called when the component is being destroyed.

Declaration: `procedure Destroying`

Visibility: `public`

Description: Destroying sets the `csDestroying` flag in the component's state (386) property, and does the same for all owned components.

It is not necessary to call Destroying directly, the destructor Destroy (388) does this automatically.

See also: State (386), Destroy (388)

4.47.12 TComponent.ExecuteAction

Synopsis: Standard action execution method.

Declaration: `function ExecuteAction(Action: TBasicAction) : Boolean; Dynamic`

Visibility: `public`

Description: ExecuteAction checks whether Action handles the current component, and if yes, calls the ExecuteAction method, passing itself as a parameter. The function returns True if the action handles the current component.

See also: TBasicAction (349), TBasicAction.ExecuteAction (349), TBasicAction.HandlesTarget (350), TComponent.UpdateAction (392)

4.47.13 TComponent.FindComponent

Synopsis: Finds and returns the named component in the owned components.

Declaration: `function FindComponent(const AName: string) : TComponent`

Visibility: public

Description: `FindComponent` searches the component with name `AName` in the list of owned components. If `AName` is empty, then `Nil` is returned.

See also: Components (393), Name (396)

4.47.14 TComponent.FreeNotification

Synopsis: Ask the component to notify called when it is being destroyed.

Declaration: `procedure FreeNotification(AComponent: TComponent)`

Visibility: public

Description: `FreeNotification` inserts `AComponent` in the freenotification list. When the component is destroyed, the `Notification` (387) method is called for all components in the freenotification list.

See also: Components (393), Notification (387), `TComponent.RemoveFreeNotification` (390)

4.47.15 TComponent.RemoveFreeNotification

Synopsis: Remove a component from the Free Notification list.

Declaration: `procedure RemoveFreeNotification(AComponent: TComponent)`

Visibility: public

Description: `RemoveFreeNotification` removes `AComponent` from the freenotification list.

See also: `TComponent.FreeNotification` (390)

4.47.16 TComponent.FreeOnRelease

Synopsis: Part of the `IVCLComObject` interface.

Declaration: `procedure FreeOnRelease`

Visibility: public

Description: Provided for Delphi compatibility, but is not yet implemented.

4.47.17 TComponent.GetEnumerator

Synopsis: Create an `IEnumerator` instance.

Declaration: `function GetEnumerator : TComponentEnumerator`

Visibility: public

Description: `GetEnumerator` is the implementation of the `IEnumerable` (1591) interface for `TComponent`. It creates a `TComponentEnumerator` (396) instance and returns its `IEnumerator` (1591) interface. The enumerator enumerates all child components of the component instance.

See also: `TComponentEnumerator` (396), `IEnumerator` (1591), `IEnumerable` (1591)

4.47.18 TComponent.GetNamePath

Synopsis: Returns the name path of this component.

Declaration: `function GetNamePath : string; Override`

Visibility: `public`

Description: `GetNamePath` returns the name of the component as it will be shown in the object inspector.

`TComponent` overrides `GetNamePath` so it returns the `Name` (396) property of the component.

See also: `Name` (396), `TPersistent.GetNamePath` (448)

4.47.19 TComponent.GetParentComponent

Synopsis: Returns the parent component.

Declaration: `function GetParentComponent : TComponent; Dynamic`

Visibility: `public`

Description: `GetParentComponent` can be implemented to return the parent component of this component.

The implementation of this method in `TComponent` always returns `Nil`. Descendant classes must override this method to return the visual parent of the component.

See also: `HasParent` (391), `Owner` (395)

4.47.20 TComponent.HasParent

Synopsis: Does the component have a parent ?

Declaration: `function HasParent : Boolean; Dynamic`

Visibility: `public`

Description: `HasParent` can be implemented to return whether the parent of the component exists. The implementation of this method in `TComponent` always returns `False`, and should be overridden by descendant classes to return `True` when a parent is available. If `HasParent` returns `True`, then `GetParentComponent` (391) will return the parent component.

See also: `HasParent` (391), `Owner` (395)

4.47.21 TComponent.InsertComponent

Synopsis: Insert the given component in the list of owned components.

Declaration: `procedure InsertComponent (AComponent : TComponent)`

Visibility: `public`

Description: `InsertComponent` attempts to insert `AComponent` in the list with owned components. It first calls `ValidateComponent` (386) to see whether the component can be inserted. It then checks whether there are no name conflicts by calling `ValidateRename` (386). If neither of these checks have raised an exception the component is inserted, and notified of the insert.

See also: `RemoveComponent` (392), `Insert` (386), `ValidateContainer` (386), `ValidateRename` (386), `Notification` (387)

4.47.22 TComponent.RemoveComponent

Synopsis: Remove the given component from the list of owned components.

Declaration: `procedure RemoveComponent (AComponent : TComponent)`

Visibility: `public`

Description: `RemoveComponent` will send an `opRemove` notification to `AComponent` and will then proceed to remove `AComponent` from the list of owned components.

See also: `InsertComponent` (391), `Remove` (386), `ValidateRename` (386), `Notification` (387)

4.47.23 TComponent.SafeCallException

Synopsis: Part of the `IVCLComObject` Interface.

Declaration: `function SafeCallException (ExceptObject : TObject;
ExceptAddr : CodePointer) : HRESULT; Override`

Visibility: `public`

Description: Provided for Delphi compatibility, but not implemented.

4.47.24 TComponent.SetSubComponent

Synopsis: Sets the `csSubComponent` style.

Declaration: `procedure SetSubComponent (ASubComponent : Boolean)`

Visibility: `public`

Description: `SetSubComponent` includes `csSubComponent` in the `ComponentStyle` (395) property if `ASubComponent` is `True`, and excludes it again if `ASubComponent` is `False`.

See also: `TComponent.ComponentStyle` (395)

4.47.25 TComponent.UpdateAction

Synopsis: Updates the state of an action.

Declaration: `function UpdateAction (Action : TBasicAction) : Boolean; Dynamic`

Visibility: `public`

Description: `UpdateAction` checks whether `Action` handles the current component, and if yes, calls the `UpdateTarget` method, passing itself as a parameter. The function returns `True` if the action handles the current component.

See also: `TBasicAction` (349), `TBasicAction.UpdateTarget` (351), `TBasicAction.HandlesTarget` (350), `TBasicAction.ExecuteAction` (349)

4.47.26 TComponent.IsImplementorOf

Synopsis: Checks if the current component is the implementor of the interface.

Declaration: `function IsImplementorOf(const Intf: IInterface) : Boolean`

Visibility: public

Description: `IsImplementorOf` returns `True` if the current component implements the given interface. The interface should descend from `IInterfaceComponentReference` (325) and the `GetComponent` method should return the current instance.

See also: `IInterfaceComponentReference` (325)

4.47.27 TComponent.ReferenceInterface

Synopsis: Interface implementation of Notification.

Declaration: `procedure ReferenceInterface(const intf: IInterface; op: TOperation)`

Visibility: public

Description: `ReferenceInterface` can be used to notify an interface of a component operation: it is the equivalent of the `TComponent.Notification` (387) method of `TComponent` for interfaces. If the interface implements `IInterfaceComponentReference` (325), then the component that implements the interface is notified of the given operation `Op`.

Errors: None.

See also: `TComponent.Notification` (387), `IInterfaceComponentReference` (325)

4.47.28 TComponent.ComObject

Synopsis: Interface reference implemented by the component.

Declaration: `Property ComObject : IUnknown`

Visibility: public

Access: Read

Description: `ComObject` returns the COM interface represented by the component. If the component does not represent a COM interface, reading this property will raise an `EComponentError` (319).

See also: `EComponentError` (319)

4.47.29 TComponent.Components

Synopsis: Indexed list (zero-based) of all owned components.

Declaration: `Property Components[Index: Integer]: TComponent`

Visibility: public

Access: Read

Description: `Components` provides indexed access to the list of owned components. `Index` can range from 0 to `ComponentCount-1` (394).

See also: `ComponentCount` (394), `Owner` (395)

4.47.30 TComponent.ComponentCount

Synopsis: Count of owned components.

Declaration: `Property ComponentCount : Integer`

Visibility: `public`

Access: `Read`

Description: `ComponentCount` returns the number of components that the current component owns. It can be used to determine the valid index range in the `Component` (393) array.

See also: `Components` (393), `Owner` (395)

4.47.31 TComponent.ComponentIndex

Synopsis: Index of component in it's owner's list.

Declaration: `Property ComponentIndex : Integer`

Visibility: `public`

Access: `Read, Write`

Description: `ComponentIndex` is the index of the current component in its owner's list of components. If the component has no owner, the value of this property is -1.

See also: `Components` (393), `ComponentCount` (394), `Owner` (395)

4.47.32 TComponent.ComponentState

Synopsis: Current component's state.

Declaration: `Property ComponentState : TComponentState`

Visibility: `public`

Access: `Read`

Description: `ComponentState` indicates the current state of the component. It is a set of flags which indicate the various stages in the lifetime of a component. The following values can occur in this set:

Table 4.25: Component states

Flag	Meaning
<code>csLoading</code>	The component is being loaded from stream
<code>csReading</code>	Component properties are being read from stream.
<code>csWriting</code>	Component properties are being written to stream.
<code>csDestroying</code>	The component or one of it's owners is being destroyed.
<code>csAncestor</code>	The component is being streamed as part of a frame
<code>csUpdating</code>	The component is being updated
<code>csFixups</code>	References to other components are being resolved
<code>csFreeNotification</code>	The component has freenotifications.
<code>csInline</code>	The component is being loaded as part of a frame
<code>csDesignInstance</code>	? not used.

The component state is set by various actions such as reading it from stream, destroying it etc.

See also: [SetAncestor \(386\)](#), [SetDesigning \(386\)](#), [SetInline \(386\)](#), [SetDesignInstance \(386\)](#), [Updating \(386\)](#), [Updated \(386\)](#), [Loaded \(386\)](#)

4.47.33 TComponent.ComponentStyle

Synopsis: Current component's style.

Declaration: `Property ComponentStyle : TComponentStyle`

Visibility: public

Access: Read

Description: Current component's style.

4.47.34 TComponent.DesignInfo

Synopsis: Information for IDE designer.

Declaration: `Property DesignInfo : LongInt`

Visibility: public

Access: Read,Write

Description: `DesignInformation` can be used by an IDE to store design information in the component. It should not be used by an application programmer.

See also: [Tag \(396\)](#)

4.47.35 TComponent.Owner

Synopsis: Owner of this component.

Declaration: `Property Owner : TComponent`

Visibility: public

Access: Read

Description: `Owner` returns the owner of this component. The owner cannot be set except by explicitly inserting the component in another component's owned components list using that component's [InsertComponent \(391\)](#) method, or by removing the component from it's owner's owned component list using the [RemoveComponent \(392\)](#) method.

See also: [Components \(393\)](#), [InsertComponent \(391\)](#), [RemoveComponent \(392\)](#)

4.47.36 TComponent.VCLComObject

Synopsis: Not implemented.

Declaration: `Property VCLComObject : Pointer`

Visibility: public

Access: Read,Write

Description: `VCLComObject` is not yet implemented in Free Pascal.

4.47.37 TComponent.Name

Synopsis: Name of the component.

Declaration: `Property Name : TComponentName`

Visibility: published

Access: Read,Write

Description: `Name` is the name of the component. This name should be a valid identifier, i.e. must start with a letter or underscore, and can contain only letters, numbers and the underscore character. When attempting to set the name of a component, the name will be checked for validity. Furthermore, when a component is owned by another component, the name must be either empty or must be unique among the child component names.

By "letters", 7-bit letters are meant.

Errors: Attempting to set the name to an invalid value will result in an exception being raised.

See also: `ValidateRename` ([386](#)), `Owner` ([395](#))

4.47.38 TComponent.Tag

Synopsis: Tag value of the component.

Declaration: `Property Tag : PtrInt`

Visibility: published

Access: Read,Write

Description: `Tag` can be used to store an integer value in the component. This value is streamed together with all other published properties. It can be used for instance to quickly identify a component in an event handler.

See also: `Name` ([396](#))

4.48 TComponentEnumerator

4.48.1 Description

`TComponentEnumerator` implements the `#rtl.system.IEnumerator` ([1591](#)) interface for the `TComponent` ([386](#)) class, so the `TComponent` class can be used in a `for ... in` loop over the `TComponent.Components` ([393](#)) child components of the component. It is returned by the `TComponent.GetEnumerator` ([390](#)) method of `TComponent`.

See also: `TComponent` ([386](#)), `TComponent.GetEnumerator` ([390](#)), `#rtl.system.IEnumerator` ([1591](#))

4.48.2 Method overview

Page	Method	Description
397	<code>Create</code>	Initialize a new instance of <code>TComponentEnumerator</code> .
397	<code>GetCurrent</code>	Return the current pointer in the list.
397	<code>MoveNext</code>	Move the position of the enumerator to the next position in the children of the component.

4.48.3 Property overview

Page	Properties	Access	Description
397	Current	r	Current pointer in the list.

4.48.4 TComponentEnumerator.Create

Synopsis: Initialize a new instance of `TComponentEnumerator`.

Declaration: `constructor Create (AComponent: TComponent)`

Visibility: `public`

Description: `Create` initializes a new instance of `TComponentEnumerator` and keeps a reference to the component `AComponent` that will be enumerated.

See also: `TComponent` ([386](#))

4.48.5 TComponentEnumerator.GetCurrent

Synopsis: Return the current pointer in the list.

Declaration: `function GetCurrent : TComponent`

Visibility: `public`

Description: `GetCurrent` returns the current `TComponent` ([386](#)) child component instance in the enumerator.

Errors: No checking is done on the validity of the current position.

See also: `MoveNext` ([397](#)), `TComponent.Components` ([393](#))

4.48.6 TComponentEnumerator.MoveNext

Synopsis: Move the position of the enumerator to the next position in the children of the component.

Declaration: `function MoveNext : Boolean`

Visibility: `public`

Description: `MoveNext` puts the pointer on the next child in the components child components, and returns `True` if this succeeded, or `False` if the pointer is past the last child in the list.

Errors: Note that if `False` is returned, calling `GetCurrent` will result in an exception.

See also: `GetCurrent` ([397](#))

4.48.7 TComponentEnumerator.Current

Synopsis: Current pointer in the list.

Declaration: `Property Current : TComponent`

Visibility: `public`

Access: `Read`

Description: `Current` redefines `GetCurrent` ([397](#)) as a property.

See also: `GetCurrent` ([397](#))

4.49 TCustomMemoryStream

4.49.1 Description

TCustomMemoryStream is the parent class for streams that stored their data in memory. It introduces all needed functions to handle reading from and navigating through the memory, and introduces a Memory (400) property which points to the memory area where the stream data is kept.

The only thing which TCustomMemoryStream does not do is obtain memory to store data when writing data or the writing of data. This functionality is implemented in descendant streams such as TMemoryStream (436). The reason for this approach is that this way it is possible to create e.g. read-only descendants of TCustomMemoryStream that point to a fixed part in memory which can be read from, but not written to.

Remark Since TCustomMemoryStream is an abstract class, do not create instances of TMemoryStream directly. Instead, create instances of descendants such as TMemoryStream (436).

See also: TMemoryStream (436), TStream (466)

4.49.2 Method overview

Page	Method	Description
398	Read	Reads Count bytes from the stream into buffer.
399	SaveToFile	Writes the contents of the stream to a file.
399	SaveToStream	Writes the contents of the memory stream to another stream.
398	Seek	Sets a new position in the stream.

4.49.3 Property overview

Page	Properties	Access	Description
400	Memory	r	Pointer to the data kept in the memory stream.

4.49.4 TCustomMemoryStream.Read

Synopsis: Reads Count bytes from the stream into buffer.

Declaration: `function Read(var Buffer; Count: LongInt) : LongInt; Override`

Visibility: public

Description: Read reads Count bytes from the stream into the memory pointed to by buffer. It returns the number of bytes actually read.

This method overrides the TStream.Read (467) method of TStream (466). It will read as much bytes as are still available in the memory area pointer to by Memory (400). After the bytes are read, the internal stream position is updated.

See also: TCustomMemoryStream.Memory (400), TStream.Read (467)

4.49.5 TCustomMemoryStream.Seek

Synopsis: Sets a new position in the stream.

Declaration: `function Seek(const Offset: Int64; Origin: TSeekOrigin) : Int64; Override`

Visibility: public

Description: `Seek` overrides the abstract `TStream.Seek` (468) method. It simply updates the internal stream position, and returns the new position.

Errors: No checking is done whether the new position is still a valid position, i.e. whether the position is still within the range `0..Size`. Attempting a seek outside the valid memory range of the stream may result in an exception at the next read or write operation.

See also: `TStream.Position` (475), `TStream.Size` (476), `TCustomMemoryStream.Memory` (400)

4.49.6 TCustomMemoryStream.SaveToStream

Synopsis: Writes the contents of the memory stream to another stream.

Declaration: `procedure SaveToStream(Stream: TStream)`

Visibility: public

Description: `SaveToStream` writes the contents of the memory stream to `Stream`. The content of `Stream` is not cleared first. The current position of the memory stream is not changed by this action.

Remark This method will work much faster than the use of the `TStream.CopyFrom` (470) method:

```
Seek(0, soFromBeginning);
Stream.CopyFrom(Self, Size);
```

because the `CopyFrom` method copies the contents in blocks, while `SaveToStream` writes the contents of the memory as one big block.

Errors: If an error occurs when writing to `Stream` an `EStreamError` (321) exception will be raised.

See also: `TCustomMemoryStream.SaveToFile` (399), `TStream.CopyFrom` (470)

4.49.7 TCustomMemoryStream.SaveToFile

Synopsis: Writes the contents of the stream to a file.

Declaration: `procedure SaveToFile(const FileName: string)`

Visibility: public

Description: `SaveToFile` writes the contents of the stream to a file with name `FileName`. It simply creates a filestream and writes the contents of the memorystream to this file stream using `TCustomMemoryStream.SaveToStream` (399).

Remark This method will work much faster than the use of the `TStream.CopyFrom` (470) method:

```
Stream:=TFileStream.Create(fmCreate, FileName);
Seek(0, soFromBeginning);
Stream.CopyFrom(Self, Size);
```

because the `CopyFrom` method copies the contents in blocks, while `SaveToFile` writes the contents of the memory as one big block.

Errors: If an error occurs when creating or writing to the file, an `EStreamError` (321) exception may occur.

See also: `TCustomMemoryStream.SaveToStream` (399), `TFileStream` (406), `TStream.CopyFrom` (470)

4.49.8 TCustomMemoryStream.Memory

Synopsis: Pointer to the data kept in the memory stream.

Declaration: `Property Memory : Pointer`

Visibility: `public`

Access: `Read`

Description: `Memory` points to the memory area where stream keeps it's data. The property is read-only, so the pointer cannot be set this way.

Remark Do not write to the memory pointed to by `Memory`, since the memory content may be read-only, and thus writing to it may cause errors.

See also: `TStream.Size` ([476](#))

4.50 TDataModule

4.50.1 Description

`TDataModule` is a container for non-visual objects which can be used in an IDE to group non-visual objects which can be used by various other containers (forms) in a project. Notably, data access components are typically stored on a datamodule. Web components and services can also be implemented as descendants of datamodules.

`TDataModule` introduces some events which make it easier to program, and provides the needed streaming capabilities for persistent storage.

An IDE will typically allow to create a descendant of `TDataModule` which contains non-visual components in it's published property list.

See also: `TDataModule.OnCreate` ([403](#))

4.50.2 Method overview

Page	Method	Description
401	<code>AfterConstruction</code>	Overrides standard <code>TObject</code> (1597) behaviour.
402	<code>BeforeDestruction</code>	
401	<code>Create</code>	Create a new instance of a <code>TDataModule</code> .
401	<code>CreateNew</code>	
401	<code>Destroy</code>	Destroys the <code>TDataModule</code> instance.

4.50.3 Property overview

Page	Properties	Access	Description
402	<code>DesignOffset</code>	rw	Position property needed for manipulation in an IDE.
402	<code>DesignPPI</code>	rw	Design Time PPI.
402	<code>DesignSize</code>	rw	Size property needed for manipulation in an IDE.
403	<code>OldCreateOrder</code>	rw	Determines when <code>OnCreate</code> and <code>OnDestroy</code> are triggered.
403	<code>OnCreate</code>	rw	Event handler, called when the datamodule is created.
403	<code>OnDestroy</code>	rw	Event handler, called when the datamodule is destroyed.

4.50.4 TDataModule.Create

Synopsis: Create a new instance of a TDataModule.

Declaration: `constructor Create(AOwner: TComponent); Override`

Visibility: `public`

Description: `Create` creates a new instance of the TDataModule and calls TDataModule.CreateNew (401). After that it reads the published properties from a stream using InitInheritedComponent (309) if a descendant class is instantiated. If the OldCreateOrder (403) property is `True`, the TDataModule.OnCreate (403) event is called.

Errors: An exception can be raised during the streaming operation.

See also: TDataModule.CreateNew (401)

4.50.5 TDataModule.CreateNew

Synopsis:

Declaration: `constructor CreateNew(AOwner: TComponent)`
`constructor CreateNew(AOwner: TComponent; CreateMode: Integer); Virtual`

Visibility: `public`

Description: `CreateNew` creates a new instance of the class, but bypasses the streaming mechanism. The `CreateMode` parameter (by default zero) is not used in TDataModule. If the AddDataModule (302) handler is set, then it is called, with the newly created instance as an argument.

See also: TDataModule.Create (401), AddDataModule (302), TDataModule.OnCreate (403)

4.50.6 TDataModule.Destroy

Synopsis: Destroys the TDataModule instance.

Declaration: `destructor Destroy; Override`

Visibility: `public`

Description: `Destroy` destroys the TDataModule instance. If the OldCreateOrder (403) property is `True` the OnDestroy (403) event handler is called prior to destroying the data module.

Before calling the inherited destroy, the RemoveDataModule (303) handler is called if it is set, and `Self` is passed as a parameter.

Errors: An event can be raised during the OnDestroy event handler.

See also: TDataModule.OnDestroy (403), RemoveDataModule (303)

4.50.7 TDataModule.AfterConstruction

Synopsis: Overrides standard TObject (1597) behaviour.

Declaration: `procedure AfterConstruction; Override`

Visibility: `public`

Description: `AfterConstruction` calls the OnCreate (403) handler if the OldCreateOrder (403) property is `False`.

See also: TDataModule.OldCreateOrder (403), TDataModule.OnCreate (403)

4.50.8 TDataModule.BeforeDestruction

Synopsis:

Declaration: `procedure BeforeDestruction; Override`

Visibility: `public`

Description: `BeforeDestruction` calls the `OnDestroy` (403) handler if the `OldCreateOrder` (403) property is `False`.

See also: `TDataModule.OldCreateOrder` (403), `TDataModule.OnDestroy` (403)

4.50.9 TDataModule.DesignOffset

Synopsis: Position property needed for manipulation in an IDE.

Declaration: `Property DesignOffset : TPoint`

Visibility: `public`

Access: `Read,Write`

Description: `DesignOffset` is the position of the datamodule when displayed in an IDE. It is streamed to the form file, and should not be used at run-time.

See also: `TDataModule.DesignSize` (402)

4.50.10 TDataModule.DesignSize

Synopsis: Size property needed for manipulation in an IDE.

Declaration: `Property DesignSize : TPoint`

Visibility: `public`

Access: `Read,Write`

Description: `DesignSize` is the size of the datamodule when displayed in an IDE. It is streamed to the form file, and should not be used at run-time.

See also: `TDataModule.DesignOffset` (402)

4.50.11 TDataModule.DesignPPI

Synopsis: Design Time PPI.

Declaration: `Property DesignPPI : Integer`

Visibility: `public`

Access: `Read,Write`

Description: `DesignPPI` exists for the benefit of graphical designers, to store the PPI at which the module was designed. It is not used in FPC itself.

4.50.12 TDataModule.OnCreate

Synopsis: Event handler, called when the datamodule is created.

Declaration: `Property OnCreate : TNotifyEvent`

Visibility: published

Access: Read,Write

Description: The `OnCreate` event is triggered when the datamodule is created and streamed. The exact moment of triggering is dependent on the value of the `OldCreateOrder` (403) property.

See also: `TDataModule.Create` (401), `TDataModule.CreateNew` (401), `TDataModule.OldCreateOrder` (403)

4.50.13 TDataModule.OnDestroy

Synopsis: Event handler, called when the datamodule is destroyed.

Declaration: `Property OnDestroy : TNotifyEvent`

Visibility: published

Access: Read,Write

Description: The `OnDestroy` event is triggered when the datamodule is destroyed. The exact moment of triggering is dependent on the value of the `OldCreateOrder` (403) property.

See also: `TDataModule.Destroy` (401), `TDataModule.OnCreate` (403), `TDataModule.Create` (401), `TDataModule.CreateNew` (401), `TDataModule.OldCreateOrder` (403)

4.50.14 TDataModule.OldCreateOrder

Synopsis: Determines when `OnCreate` and `OnDestroy` are triggered.

Declaration: `Property OldCreateOrder : Boolean`

Visibility: published

Access: Read,Write

Description: `OldCreateOrder` determines when exactly the `OnCreate` (403) and `OnDestroy` (403) event handlers are called.

If set to `True`, then the `OnCreate` event handler is called after the data module was streamed. If it is set to `False`, then the handler is called prior to the streaming process.

If set to `True`, then the `OnDestroy` event handler is called before the data module is removed from the streaming system. If it is set to `False`, then the handler is called after the data module was removed from the streaming process.

See also: `TDataModule.OnDestroy` (403), `TDataModule.OnCreate` (403), `TDataModule.Destroy` (401), `TDataModule.Create` (401), `TDataModule.CreateNew` (401), `TDataModule.OldCreateOrder` (403)

4.51 TFiler

4.51.1 Description

Class responsible for streaming of components.

4.51.2 Method overview

Page	Method	Description
404	DefineBinaryProperty	
404	DefineProperty	
404	FlushBuffer	Flush the buffer.

4.51.3 Property overview

Page	Properties	Access	Description
405	Ancestor	rw	Ancestor component from which an inherited component is streamed.
405	IgnoreChildren	rw	Determines whether children will be streamed as well.
405	LookupRoot	r	Component used to look up ancestor components.
405	Root	rw	The root component is the initial component which is being streamed.

4.51.4 TFiler.DefineProperty

Synopsis:

Declaration: `procedure DefineProperty(const Name: string; ReadData: TReaderProc;
WriteData: TWriterProc; HasData: Boolean)
; Virtual; Abstract`

Visibility: public

Description:

4.51.5 TFiler.DefineBinaryProperty

Synopsis:

Declaration: `procedure DefineBinaryProperty(const Name: string;
ReadData: TStreamProc;
WriteData: TStreamProc; HasData: Boolean)
; Virtual; Abstract`

Visibility: public

Description:

4.51.6 TFiler.FlushBuffer

Synopsis: Flush the buffer.

Declaration: `procedure FlushBuffer; Virtual; Abstract`

Visibility: public

Description: `FlushBuffer` flushes the buffer by calling the `flushbuffer` on the driver. It is provided for Delphi compatibility, and is not used in FPC.

See also: `TAbstractObjectReader.FlushBuffer` ([336](#))

4.51.7 TFile.Root

Synopsis: The root component is the initial component which is being streamed.

Declaration: `Property Root : TComponent`

Visibility: `public`

Access: `Read,Write`

Description: The streaming process will stream a component and all the components which it owns. The `Root` component is the component which is initially streamed.

See also: `LookupRoot` ([405](#))

4.51.8 TFile.LookupRoot

Synopsis: Component used to look up ancestor components.

Declaration: `Property LookupRoot : TComponent`

Visibility: `public`

Access: `Read`

Description: When comparing inherited component's values against parent values, the values are compared with the component in `LookupRoot`. Initially, it is set to `Root` ([405](#)).

See also: `Root` ([405](#))

4.51.9 TFile.Ancestor

Synopsis: Ancestor component from which an inherited component is streamed.

Declaration: `Property Ancestor : TPersistent`

Visibility: `public`

Access: `Read,Write`

Description: When streaming a component, this is the parent component. Only properties that differ from the parent's property value will be streamed.

See also: `Root` ([405](#)), `LookupRoot` ([405](#))

4.51.10 TFile.IgnoreChildren

Synopsis: Determines whether children will be streamed as well.

Declaration: `Property IgnoreChildren : Boolean`

Visibility: `public`

Access: `Read,Write`

Description: By default, all children (i.e. owned objects) will also be streamed when streaming a component. This property can be used to prevent owned objects from being streamed.

4.52 TFileStream

4.52.1 Description

TFileStream is a TStream (466) descendant that stores or reads its data from a named file in the file system of the operating system.

To this end, it overrides some of the methods in `TStream` and implements them for the case of files on disk, and it adds the `FileName` (407) property to the list of public properties.

See also: [TFileStream.Create \(406\)](#), [TStream \(466\)](#)

4.52.2 Method overview

Page	Method	Description
406	Create	Creates a file stream.
407	Destroy	Destroys the file stream.
407	Flush	

4.52.3 Property overview

Page	Properties	Access	Description
407	FileName	r	The filename of the stream.

4.52.4 TFileStream.Create

Synopsis: Creates a file stream.

```
Declaration: constructor Create(const AFileName: string; Mode: Word)
              constructor Create(const AFileName: string; Mode: Word;
                                Rights: Cardinal)
```

Visibility: public

Description: `Create` creates a new instance of a `TFileStream` class. It opens the file `AFileName` with mode `Mode`, which can have one of the following values:

Table 4.26:

fmCreate	TFileStream.Create (406) creates a new file if needed.
fmOpenRead	TFileStream.Create (406) opens a file with read-only access.
fmOpenWrite	TFileStream.Create (406) opens a file with write-only access.
fmOpenReadWrite	TFileStream.Create (406) opens a file with read-write access.

These constants (except `fmCreate`) can be OR-ed with the following to specify how sharing and file locking is supposed to be handled:

Table 4.27:

<code>fmShareCompat</code>	Open file in DOS share-compatibility mode.
<code>fmShareExclusive</code>	Lock file for exclusive use.
<code>fmShareDenyWrite</code>	Lock file so other processes can only read.
<code>fmShareDenyRead</code>	Lock file so other processes cannot read.
<code>fmShareDenyNone</code>	Do not lock file.

Note that sharing is advisory on Unix-like platforms.

After the file has been opened in the requested mode and a handle has been obtained from the operating system, the inherited constructor is called.

Errors: If the file could not be opened in the requested mode, an `EFOpenError` (320) exception is raised.

See also: `TStream` (466), `TFileStream.FileName` (407), `THandleStream.Create` (417)

4.52.5 TFileStream.Destroy

Synopsis: Destroys the file stream.

Declaration: `destructor Destroy; Override`

Visibility: `public`

Description: `Destroy` closes the file (causing possible buffered data to be written to disk) and then calls the inherited destructor.

Do not call `destroy` directly, instead call the `Free` method. `Destroy` does not check whether `Self` is `nil`, while `Free` does.

See also: `TFileStream.Create` (406)

4.52.6 TFileStream.Flush

Declaration: `function Flush : Boolean`

Visibility: `public`

4.52.7 TFileStream.FileName

Synopsis: The filename of the stream.

Declaration: `Property FileName : string`

Visibility: `public`

Access: `Read`

Description: `FileName` is the name of the file that the stream reads from or writes to. It is the name as passed in the constructor of the stream; it cannot be changed. To write to another file, the stream must be freed and created again with the new filename.

See also: `TFileStream.Create` (406)

4.53 TFPList

4.53.1 Description

`TFPList` is a class that can be used to manage collections of pointers. It introduces methods and properties to store the pointers, search in the list of pointers, sort them. It manages its memory by itself, no intervention for that is needed. Contrary to `TList` (426), `TFPList` has no notification mechanism. If no notification mechanism is used, it is better to use `TFPList` instead of `TList`, as the performance of `TFPList` is much higher.

To manage collections of strings, it is better to use a `TStrings` (486) descendant such as `TStringList` (481). To manage general objects, a `TCollection` (376) class exists, from which a descendant can be made to manage collections of various kinds.

See also: `TStrings` (486), `TCollection` (376)

4.53.2 Method overview

Page	Method	Description
409	Add	Adds a new pointer to the list.
409	AddList	Add all pointers from another list.
413	Assign	Assign performs the given operation on the list.
409	Clear	Clears the pointer list.
411	Contains	
410	Delete	Removes a pointer from the list.
409	Destroy	Destroys the list and releases the memory used to store the list elements.
410	Error	Raises an <code>EListError</code> (284) exception.
410	Exchange	Exchanges two pointers in the list.
410	Expand	Increases the capacity of the list if needed.
411	Extract	Remove the first occurrence of a pointer from the list.
411	First	Returns the first non-nil pointer in the list.
414	ForEachCall	Call a procedure or method for each pointer in the list.
411	GetEnumerator	Create an <code>IEnumerator</code> instance.
411	IndexOf	Returns the index of a given pointer.
412	IndexOfItem	Search an item in the list.
412	Insert	Inserts a new pointer in the list at a given position.
412	Last	Returns the last non-nil pointer in the list.
412	Move	Moves a pointer from one position in the list to another.
413	Pack	Removes <code>Nil</code> pointers from the list and frees unused memory.
413	Remove	Removes a value from the list.
413	Sort	Sorts the pointers in the list.

4.53.3 Property overview

Page	Properties	Access	Description
414	Capacity	rw	Current capacity (i.e. number of pointers that can be stored) of the list.
414	Count	rw	Current number of pointers in the list.
415	Items	rw	Provides access to the pointers in the list.
415	List	r	Memory array where pointers are stored.

4.53.4 TFPList.Destroy

Synopsis: Destroys the list and releases the memory used to store the list elements.

Declaration: `destructor Destroy; Override`

Visibility: `public`

Description: `Destroy` destroys the list and releases the memory used to store the list elements. The elements themselves are in no way touched, i.e. any memory they point to must be explicitly released before calling the destructor.

4.53.5 TFPList.AddList

Synopsis: Add all pointers from another list.

Declaration: `procedure AddList (AList: TFPList)`

Visibility: `public`

Description: `AddList` adds all pointers from `AList` to the list. If a pointer is already present, it is added a second time.

See also: `TFPList.Assign` ([413](#)), `TList.AddList` ([429](#))

4.53.6 TFPList.Add

Synopsis: Adds a new pointer to the list.

Declaration: `function Add (Item: Pointer) : Integer`

Visibility: `public`

Description: `Add` adds a new pointer to the list after the last pointer (i.e. at position `Count`, thus increasing the item count with 1. If the list is at full capacity, the capacity of the list is expanded, using the `Expand` ([410](#)) method.

To insert a pointer at a certain position in the list, use the `Insert` ([412](#)) method instead.

See also: `Delete` ([410](#)), `Grow` ([426](#)), `Insert` ([412](#))

4.53.7 TFPList.Clear

Synopsis: Clears the pointer list.

Declaration: `procedure Clear`

Visibility: `public`

Description: `Clear` removes all pointers from the list, and sets the capacity to 0, thus freeing any memory allocated to maintain the list.

See also: `Destroy` ([409](#))

4.53.8 TFPList.Delete

Synopsis: Removes a pointer from the list.

Declaration: `procedure Delete(Index: Integer)`

Visibility: `public`

Description: `Delete` removes the pointer at position `Index` from the list, shifting all following pointers one position up (or to the left).

The memory the pointer is pointing to is *not* deallocated.

4.53.9 TFPList.Error

Synopsis: Raises an `EListError` (284) exception.

Declaration: `class procedure Error(const Msg: string; Data: PtrInt)`

Visibility: `public`

Description: `Error` raises an `EListError` (284) exception, with a message formatted with `Msg` and `Data`.

4.53.10 TFPList.Exchange

Synopsis: Exchanges two pointers in the list.

Declaration: `procedure Exchange(Index1: Integer; Index2: Integer)`

Visibility: `public`

Description: `Exchange` exchanges the pointers at positions `Index1` and `Index2`. Both pointers must be within the current range of the list, or an `EListError` (284) exception will be raised.

4.53.11 TFPList.Expand

Synopsis: Increases the capacity of the list if needed.

Declaration: `function Expand : TFPList`

Visibility: `public`

Description: `Expand` increases the capacity of the list if the current element count matches the current list capacity.

The capacity is increased according to the following algorithm:

- 1.If the capacity is less than 3, the capacity is increased with 4.
- 2.If the capacity is larger than 3 and less than 8, the capacity is increased with 8.
- 3.If the capacity is larger than 8, the capacity is increased with 16.

The return value is `Self`.

See also: `Capacity` (414)

4.53.12 TFPList.Extract

Synopsis: Remove the first occurrence of a pointer from the list.

Declaration: `function Extract (Item: Pointer) : Pointer`

Visibility: public

Description: `Extract` searches for the first occurrence of `Item` in the list and deletes it from the list. If `Item` was found, it's value is returned. If `Item` was not found, `Nil` is returned.

See also: `TFPList.Delete` ([410](#))

4.53.13 TFPList.First

Synopsis: Returns the first non-nil pointer in the list.

Declaration: `function First : Pointer`

Visibility: public

Description: `First` returns the value of the first non-nil pointer in the list.

If there are no pointers in the list or all pointers equal `Nil`, then `Nil` is returned.

See also: `Last` ([412](#))

4.53.14 TFPList.GetEnumerator

Synopsis: Create an `IEnumerator` instance.

Declaration: `function GetEnumerator : TFPListEnumerator`

Visibility: public

Description: `GetEnumerator` is the implementation of the `IEnumerable` ([1591](#)) interface for `TFPList`. It creates a `TFPListEnumerator` ([415](#)) instance and returns it's `IEnumerator` ([1591](#)) interface.

See also: `TFPListEnumerator` ([415](#)), `IEnumerator` ([1591](#)), `IEnumerable` ([1591](#))

4.53.15 TFPList.Contains

Declaration: `function Contains (Item: Pointer) : Boolean`

Visibility: public

4.53.16 TFPList.IndexOf

Synopsis: Returns the index of a given pointer.

Declaration: `function IndexOf (Item: Pointer) : Integer`

Visibility: public

Description: `IndexOf` searches for the pointer `Item` in the list of pointers, and returns the index of the pointer, if found.

If no pointer with the value `Item` was found, -1 is returned.

4.53.17 TFPList.IndexOfItem

Synopsis: Search an item in the list.

Declaration: `function IndexOfItem(Item: Pointer; Direction: TDirection) : Integer`

Visibility: public

Description: `IndexOfItem` has the same function as the `IndexOf` (281) function: it searches for `Item` in the list, and returns the index of the first found matching pointer. If none is found, -1 is returned.

In difference with the `IndexOf` function, it accepts a parameter `Direction` indicating the search direction: from the beginning of the list till the end of the list, or from the end of the list till the beginning. The `IndexOf` function starts at the beginning of the list. The search direction is only important if the item can appear multiple times in the list.

See also: `TFPList.TDirection` (??), `TFPList.IndexOf` (411)

4.53.18 TFPList.Insert

Synopsis: Inserts a new pointer in the list at a given position.

Declaration: `procedure Insert(Index: Integer; Item: Pointer)`

Visibility: public

Description: `Insert` inserts pointer `Item` at position `Index` in the list. All pointers starting from `Index` are shifted to the right.

If `Index` is not a valid position, then a `EListError` (284) exception is raised.

See also: `Add` (409), `Delete` (410)

4.53.19 TFPList.Last

Synopsis: Returns the last non-nil pointer in the list.

Declaration: `function Last : Pointer`

Visibility: public

Description: `Last` returns the value of the last non-nil pointer in the list.

If there are no pointers in the list or all pointers equal `Nil`, then `Nil` is returned.

See also: `First` (411)

4.53.20 TFPList.Move

Synopsis: Moves a pointer from one position in the list to another.

Declaration: `procedure Move(CurIndex: Integer; NewIndex: Integer)`

Visibility: public

Description: `Move` moves the pointer at position `CurIndex` to position `NewIndex`. This is done by storing the value at position `CurIndex`, deleting the pointer at position `CurIndex`, and reinserting the value at position `NewIndex`.

If `CurIndex` or `Newindex` are not inside the valid range of indices, an `EListError` (284) exception is raised.

See also: `Exchange` (410)

4.53.21 TFPList.Assign

Synopsis: Assign performs the given operation on the list.

Declaration: `procedure Assign(ListA: TFPList; AOperator: TListAssignOp;
ListB: TFPList)`

Visibility: public

Description: Assign can be used to merge or assign lists. It is an extended version of the usual `TPersistent.Assign` mechanism. The arguments `ListA` and `ListB` are used as sources of pointers to add or remove elements from the current list, depending on the operation `AOperation`. The available operations are documented in the `TListAssignOp` (291) type.

See also: `TFPList.Add` (409), `TFPList.Clear` (409)

4.53.22 TFPList.Remove

Synopsis: Removes a value from the list.

Declaration: `function Remove(Item: Pointer) : Integer`

Visibility: public

Description: Remove searches `Item` in the list, and, if it finds it, deletes the item from the list. Only the first occurrence of `Item` is removed.

See also: `Delete` (410), `IndexOf` (411), `Insert` (412)

4.53.23 TFPList.Pack

Synopsis: Removes `Nil` pointers from the list and frees unused memory.

Declaration: `procedure Pack`

Visibility: public

Description: Pack removes all `nil` pointers from the list. The capacity of the list is then set to the number of pointers in the list. This method can be used to free unused memory if the list has grown to very large sizes and has a lot of unneeded `nil` pointers in it.

See also: `TFPList.Clear` (409)

4.53.24 TFPList.Sort

Synopsis: Sorts the pointers in the list.

Declaration: `procedure Sort(Compare: TListSortCompare)`

Visibility: public

Description: `Sort` sorts the pointers in the list. Two pointers are compared by passing them to the `Compare` function. The result of this function determines how the pointers will be sorted:

- If the result of this function is negative, the first pointer is assumed to be 'less' than the second and will be moved before the second in the list.
- If the function result is positive, the first pointer is assumed to be 'greater than' the second and will be moved after the second in the list.

- if the function result is zero, the pointers are assumed to be 'equal' and no moving will take place.

The sort is done using a quicksort algorithm.

4.53.25 TFPList.ForEachCall

Synopsis: Call a procedure or method for each pointer in the list.

Declaration: `procedure ForEachCall(proc2call: TListCallback; arg: pointer)`
`procedure ForEachCall(proc2call: TListStaticCallback; arg: pointer)`

Visibility: public

Description: `ForEachCall` iterates over all pointers in the list and calls `proc2call`, passing it the pointer and the additional `arg` data pointer. `Proc2Call` can be a method or a static procedure.

Errors: None.

See also: `TListStaticCallback` ([292](#)), `TListCallback` ([291](#))

4.53.26 TFPList.Capacity

Synopsis: Current capacity (i.e. number of pointers that can be stored) of the list.

Declaration: `Property Capacity : Integer`

Visibility: public

Access: Read,Write

Description: `Capacity` contains the number of pointers the list can store before it starts to grow.

If a new pointer is added to the list using `add` ([409](#)) or `insert` ([412](#)), and there is not enough memory to store the new pointer, then the list will try to allocate more memory to store the new pointer. Since this is a time consuming operation, it is important that this operation be performed as little as possible. If it is known how many pointers there will be before filling the list, it is a good idea to set the capacity first before filling. This ensures that the list doesn't need to grow, and will speed up filling the list.

See also: `SetCapacity` ([408](#)), `Count` ([414](#))

4.53.27 TFPList.Count

Synopsis: Current number of pointers in the list.

Declaration: `Property Count : Integer`

Visibility: public

Access: Read,Write

Description: `Count` is the current number of (possibly `Nil`) pointers in the list. Since the list is zero-based, the index of the largest pointer is `Count-1`.

4.53.28 TFPList.Items

Synopsis: Provides access to the pointers in the list.

Declaration: `Property Items[Index: Integer]: Pointer; default`

Visibility: `public`

Access: `Read, Write`

Description: `Items` is used to access the pointers in the list. It is the default property of the `TFPList` class, so it can be omitted.

The list is zero-based, so `Index` must be in the range 0 to `Count-1`.

4.53.29 TFPList.List

Synopsis: Memory array where pointers are stored.

Declaration: `Property List : PPointerList`

Visibility: `public`

Access: `Read`

Description: `List` points to the memory space where the pointers are stored. This can be used to quickly copy the list of pointers to another location.

4.54 TFPListEnumerator

4.54.1 Description

`TFPListEnumerator` implements the `#rtl.system.IEnumerator` (1591) interface for the `TFPList` (408) class, so the `TFPList` class can be used in a `for ... in` loop. It is returned by the `TFPList.GetEnumerator` (411) method of `TFPList`.

See also: `TFPList` (408), `TFPList.GetEnumerator` (411), `#rtl.system.IEnumerator` (1591)

4.54.2 Method overview

Page	Method	Description
415	<code>Create</code>	Initialize a new instance of <code>TFPListEnumerator</code> .
416	<code>GetCurrent</code>	Return the current pointer in the list.
416	<code>MoveNext</code>	Move the position of the enumerator to the next position in the list.

4.54.3 Property overview

Page	Properties	Access	Description
416	<code>Current</code>	<code>r</code>	Current pointer in the list.

4.54.4 TFPListEnumerator.Create

Synopsis: Initialize a new instance of `TFPListEnumerator`.

Declaration: `constructor Create(AList: TFPList)`

Visibility: public

Description: `Create` initializes a new instance of `TFPListEnumerator` and keeps a reference to the list `AList` that will be enumerated.

See also: `TFPList` ([408](#))

4.54.5 `TFPListEnumerator.GetCurrent`

Synopsis: Return the current pointer in the list.

Declaration: `function GetCurrent : Pointer`

Visibility: public

Description: `GetCurrent` returns the current pointer in the enumerator.

Errors: No checking is done on the validity of the current position.

See also: `MoveNext` ([416](#))

4.54.6 `TFPListEnumerator.MoveNext`

Synopsis: Move the position of the enumerator to the next position in the list.

Declaration: `function MoveNext : Boolean`

Visibility: public

Description: `MoveNext` puts the pointer on the next item in the list, and returns `True` if this succeeded, or `False` if the pointer is past the last element in the list.

Errors: Note that if `False` is returned, calling `GetCurrent` will result in an exception.

See also: `GetCurrent` ([416](#))

4.54.7 `TFPListEnumerator.Current`

Synopsis: Current pointer in the list.

Declaration: `Property Current : Pointer`

Visibility: public

Access: Read

Description: `Current` redefines `GetCurrent` ([416](#)) as a property.

See also: `GetCurrent` ([416](#))

4.55 THandleStream

4.55.1 Description

`THandleStream` is an abstract descendant of the `TStream` (466) class that provides methods for a stream to handle all reading and writing to and from a handle, provided by the underlying OS. To this end, it overrides the `Read` (417) and `Write` (418) methods of `TStream`.

Remark

- `THandleStream` does not obtain a handle from the OS by itself, it just handles reading and writing to such a handle by wrapping the system calls for reading and writing; Descendant classes should obtain a handle from the OS by themselves and pass it on in the inherited constructor.
- Contrary to Delphi, no seek is implemented for `THandleStream`, since pipes and sockets do not support this. The seek is implemented in descendant methods that support it.

See also: `TStream` (466), `TFileStream` (406)

4.55.2 Method overview

Page	Method	Description
417	<code>Create</code>	Create a handlestream from an OS Handle.
417	<code>Read</code>	Overrides standard read method.
418	<code>Seek</code>	Overrides the <code>Seek</code> method.
418	<code>Write</code>	Overrides standard write method.

4.55.3 Property overview

Page	Properties	Access	Description
418	<code>Handle</code>	<code>r</code>	The OS handle of the stream.

4.55.4 THandleStream.Create

Synopsis: Create a handlestream from an OS Handle.

Declaration: `constructor Create(AHandle: THandle)`

Visibility: `public`

Description: `Create` creates a new instance of a `THandleStream` class. It stores `AHandle` in an internal variable and then calls the inherited constructor.

See also: `TStream` (466)

4.55.5 THandleStream.Read

Synopsis: Overrides standard read method.

Declaration: `function Read(var Buffer; Count: LongInt) : LongInt; Override`

Visibility: `public`

Description: `Read` overrides the `Read` (467) method of `TStream`. It uses the `Handle` (418) property to read the `Count` bytes into `Buffer`

If no error occurs while reading, the number of bytes actually read will be returned.

Errors: If the operating system reports an error while reading from the handle, -1 is returned.

See also: `TStream.Read` (467), `THandleStream.Write` (418), `THandleStream.Handle` (418)

4.55.6 THandleStream.Write

Synopsis: Overrides standard write method.

Declaration: `function Write(const Buffer; Count: LongInt) : LongInt; Override`

Visibility: `public`

Description: `Write` overrides the `Write` (468) method of `TStream`. It uses the `Handle` (418) property to write the `Count` bytes from `Buffer`.

If no error occurs while writing, the number of bytes actually written will be returned.

Errors: If the operating system reports an error while writing to the handle, 0 is returned.

See also: `TStream.Read` (467), `THandleStream.Write` (418), `THandleStream.Handle` (418)

4.55.7 THandleStream.Seek

Synopsis: Overrides the `Seek` method.

Declaration: `function Seek(const Offset: Int64; Origin: TSeekOrigin) : Int64; Override`

Visibility: `public`

Description: `seek` uses the `FileSeek` (1703) method to position the stream on the desired position. Note that handle stream descendants (notably pipes) can override the method to prevent the seek.

4.55.8 THandleStream.Handle

Synopsis: The OS handle of the stream.

Declaration: `Property Handle : THandle`

Visibility: `public`

Access: `Read`

Description: `Handle` represents the Operating system handle to which reading and writing is done. The handle can be read only, i.e. it cannot be set after the `THandleStream` instance was created. It should be passed to the constructor `THandleStream.Create` (417)

See also: `THandleStream` (417), `THandleStream.Create` (417)

4.56 TInterfacedPersistent

4.56.1 Description

TInterfacedPersistent is a direct descendant of TPersistent (446) which implements the #rtl.system.IInterface (1372) interface. In particular, it implements the QueryInterface as a public method.

See also: IInterface (1372)

4.56.2 Interfaces overview

Page	Interfaces	Description
1372	IInterface	Basic interface for all COM based interfaces.

4.56.3 Method overview

Page	Method	Description
419	AfterConstruction	Overrides the standard AfterConstruction method.
419	QueryInterface	Implementation of IInterface.QueryInterface.

4.56.4 TInterfacedPersistent.QueryInterface

Synopsis: Implementation of IInterface.QueryInterface.

Declaration: `function QueryInterface(const IID: TGuid; out Obj) : HRESULT; Virtual`

Visibility: public

Description: QueryInterface simply calls GetInterface using the specified IID, and returns the correct values.

See also: TObject.GetInterface (1605)

4.56.5 TInterfacedPersistent.AfterConstruction

Synopsis: Overrides the standard AfterConstruction method.

Declaration: `procedure AfterConstruction; Override`

Visibility: public

Description: AfterConstruction is overridden to do some extra interface housekeeping: a reference to the IInterface interface of the owning class is obtained (if it exists).

4.57 TInterfaceList

4.57.1 Description

TInterfaceList is a standard implementation of the IInterfaceList (326) interface. It uses a TThreadList (534) instance to store the list of interfaces.

See also: IInterfaceList (326), TList (426)

4.57.2 Interfaces overview

Page	Interfaces	Description
326	<code>IInterfaceList</code>	Interface for maintaining a list of interfaces.
331	<code>IInterfaceListEx</code>	

4.57.3 Method overview

Page	Method	Description
422	<code>Add</code>	Add an interface to the list.
421	<code>Clear</code>	Removes all interfaces from the list.
420	<code>Create</code>	Create a new instance of <code>TInterfaceList</code> .
421	<code>Delete</code>	Delete an interface from the list.
420	<code>Destroy</code>	Destroys the list of interfaces.
421	<code>Exchange</code>	Exchange 2 interfaces in the list.
424	<code>Expand</code>	Expands the list.
421	<code>First</code>	Returns the first non- <code>Nil</code> element in the list.
422	<code>GetEnumerator</code>	Create an <code>IEnumerator</code> instance.
422	<code>IndexOf</code>	Returns the index of an interface.
422	<code>Insert</code>	Insert an interface to the list.
423	<code>Last</code>	Returns the last non- <code>Nil</code> element in the list.
423	<code>Lock</code>	Lock the list.
423	<code>Remove</code>	Remove an interface from the list.
423	<code>Unlock</code>	UnLocks a locked list.

4.57.4 Property overview

Page	Properties	Access	Description
424	<code>Capacity</code>	<code>rw</code>	The current capacity of the list.
424	<code>Count</code>	<code>rw</code>	The current number of elements in the list.
424	<code>Items</code>	<code>rw</code>	Array-based access to the list's items.

4.57.5 `TInterfaceList.Create`

Synopsis: Create a new instance of `TInterfaceList`.

Declaration: `constructor Create`

Visibility: `public`

Description: `Create` creates a new instance of the `TInterfaceList` class. It sets up the internal structures needed to store the list of interfaces.

See also: `Destroy` ([420](#))

4.57.6 `TInterfaceList.Destroy`

Synopsis: Destroys the list of interfaces.

Declaration: `destructor Destroy; Override`

Visibility: `public`

Description: `Destroy` first calls `Clear` ([421](#)) and then frees the `TInterfaceList` instance from memory.

Note that the `Clear` method decreases the reference count of all interfaces.

See also: [Create \(420\)](#), [Clear \(421\)](#)

4.57.7 TInterfaceList.Clear

Synopsis: Removes all interfaces from the list.

Declaration: `procedure Clear`

Visibility: `public`

Description: `Clear` is the implementation of the `IInterfaceList.Clear (328)` method. It removes all interfaces from the list. It does this by setting each element in the list to `Nil`, in this way the reference count of each interface in the list is decreased.

See also: `IInterfaceList.Clear (328)`, [Add \(422\)](#), [Destroy \(420\)](#), [TList.Clear \(429\)](#), [TFPList.Clear \(409\)](#)

4.57.8 TInterfaceList.Delete

Synopsis: Delete an interface from the list.

Declaration: `procedure Delete(index: Integer)`

Visibility: `public`

Description: `Delete` is the implementation of the `IInterfaceList.Delete (328)` method. It clears the slot first and then removes the element from the list.

See also: `IInterfaceList.Delete (328)`, `TInterfaceList.Remove (423)`, `TInterfaceList.Add (422)`, `TList.Delete (429)`, `TFPList.Delete (410)`

4.57.9 TInterfaceList.Exchange

Synopsis: Exchange 2 interfaces in the list.

Declaration: `procedure Exchange(index1: Integer; index2: Integer)`

Visibility: `public`

Description: `Exchange` is the implementation of the `IInterfaceList.Exchange (329)` method. It exchanges the position of 2 interfaces in the list.

See also: `IInterfaceList.Exchange (329)`, `TInterfaceList.Delete (421)`, `TInterfaceList.Add (422)`, `TList.Exchange (430)`, `TFPList.Exchange (410)`

4.57.10 TInterfaceList.First

Synopsis: Returns the first non-`Nil` element in the list.

Declaration: `function First : IUnknown`

Visibility: `public`

Description: `First` is the implementation of the `IInterfaceList.First (329)` method. It returns the first non-`Nil` element from the list.

See also: `IInterfaceList.First (329)`, [TList.First \(431\)](#)

4.57.11 TInterfaceList.GetEnumerator

Synopsis: Create an `IEnumerator` instance.

Declaration: `function GetEnumerator : TInterfaceListEnumerator`

Visibility: `public`

Description: `GetEnumerator` is the implementation of the `IEnumerable` (1591) interface for `TInterfaceList`. It creates a `TInterfaceListEnumerator` (425) instance and returns its `IEnumerator` (1591) interface. The enumerator enumerates all interfaces in the list.

See also: `TInterfaceListEnumerator` (425), `IEnumerator` (1591), `IEnumerable` (1591)

4.57.12 TInterfaceList.IndexOf

Synopsis: Returns the index of an interface.

Declaration: `function IndexOf(const item: IUnknown) : Integer`

Visibility: `public`

Description: `IndexOf` is the implementation of the `IInterfaceList.IndexOf` (329) method. It returns the zero-based index in the list of the indicated interface, or -1 if the index is not in the list.

See also: `IInterfaceList.IndexOf` (329), `TList.IndexOf` (431)

4.57.13 TInterfaceList.Add

Synopsis: Add an interface to the list.

Declaration: `function Add(item: IUnknown) : Integer`

Visibility: `public`

Description: `Add` is the implementation of the `IInterfaceList.Add` (329) method. It adds an interface to the list, and returns the location of the new element in the list. This operation will increment the reference count of the interface.

See also: `IInterfaceList.Add` (329), `TInterfaceList.Delete` (421), `TInterfaceList.Insert` (422), `TList.Add` (429), `TFPList.Add` (409)

4.57.14 TInterfaceList.Insert

Synopsis: Insert an interface to the list.

Declaration: `procedure Insert(i: Integer; item: IUnknown)`

Visibility: `public`

Description: `Insert` is the implementation of the `IInterfaceList.Insert` (329) method. It inserts an interface in the list at the indicated position. This operation will increment the reference count of the interface.

See also: `IInterfaceList.Insert` (329), `TInterfaceList.Delete` (421), `TInterfaceList.Add` (422), `TList.Insert` (431), `TFPList.Insert` (412)

4.57.15 TInterfaceList.Last

Synopsis: Returns the last non-`Nil` element in the list.

Declaration: `function Last : IUnknown`

Visibility: `public`

Description: `Last` is the implementation of the `IInterfaceList.Last` (330) method. It returns the last non-`Nil` element from the list.

See also: `IInterfaceList.Last` (330), `TInterfaceList.First` (421), `TList.Last` (432), `TFPList.Last` (412)

4.57.16 TInterfaceList.Remove

Synopsis: Remove an interface from the list.

Declaration: `function Remove(item: IUnknown) : Integer`

Visibility: `public`

Description: `Remove` is the implementation of the `IInterfaceList.Remove` (330) method. It removes the first occurrence of the interface from the list.

See also: `IInterfaceList.Remove` (330), `TInterfaceList.Delete` (421), `TInterfaceList.IndexOf` (422), `TList.Remove` (432), `TFPList.Remove` (413)

4.57.17 TInterfaceList.Lock

Synopsis: Lock the list.

Declaration: `procedure Lock`

Visibility: `public`

Description: `Lock` locks the list. It is the implementation of the `IInterfaceList.Lock` (330) method. It limits access to the list to the current thread.

See also: `IInterfaceList.Lock` (330), `TInterfaceList.Unlock` (423), `TThreadList.LockList` (536)

4.57.18 TInterfaceList.Unlock

Synopsis: Unlocks a locked list.

Declaration: `procedure Unlock`

Visibility: `public`

Description: `Unlock` unlocks the list. It is the implementation of the `IInterfaceList.Unlock` (330) method. After a call to unlock, the current thread releases the list for manipulation by other threads.

See also: `IInterfaceList.Unlock` (330), `TInterfaceList.Lock` (423), `TThreadList.UnlockList` (536)

4.57.19 TInterfaceList.Expand

Synopsis: Expands the list.

Declaration: `function Expand : TInterfaceList`

Visibility: `public`

Description: `Expand` calls the `expand` method from the internally used list. It returns itself.

See also: `TList.Expand` ([430](#))

4.57.20 TInterfaceList.Capacity

Synopsis: The current capacity of the list.

Declaration: `Property Capacity : Integer`

Visibility: `public`

Access: `Read,Write`

Description: `Capacity` is the number of elements that the list can contain without needing to allocate more memory.

See also: `IInterfaceList.Capacity` ([330](#)), `TInterfaceList.Count` ([424](#)), `TList.Capacity` ([433](#)), `TFPList.Capacity` ([414](#))

4.57.21 TInterfaceList.Count

Synopsis: The current number of elements in the list.

Declaration: `Property Count : Integer`

Visibility: `public`

Access: `Read,Write`

Description: `Count` is the number of elements in the list. This can include `Nil` elements. Note that the elements are zero-based, and thus are indexed from 0 to `Count-1`.

See also: `IInterfaceList.Count` ([331](#)), `TInterfaceList.Items` ([424](#)), `TInterfaceList.Capacity` ([424](#)), `TList.Count` ([434](#)), `TFPList.Count` ([414](#))

4.57.22 TInterfaceList.Items

Synopsis: Array-based access to the list's items.

Declaration: `Property Items[Index: Integer]: IUnknown; default`

Visibility: `public`

Access: `Read,Write`

Description: `Items` provides indexed access to the elements in the list. Note that the elements are zero-based, and thus are indexed from 0 to `Count-1`. The items are read-write. It is not possible to add elements to the list by accessing an element with index larger or equal to `Count` ([424](#)).

See also: `IInterfaceList.Items` ([331](#)), `TInterfaceList.Count` ([424](#)), `TList.Items` ([434](#)), `TFPList.Items` ([415](#))

4.58 TInterfaceListEnumerator

4.58.1 Description

`TInterfaceListEnumerator` implements the `#rtl.system.IEnumerator` (1591) interface for the `TInterfaceList` (419) class, so the `TInterfaceList` class can be used in a `for ... in` loop over the `TInterfaceList.Components` (419) child components of the component. It is returned by the `TInterfaceList.GetEnumerator` (422) method of `TInterfaceList`.

See also: `TInterfaceList` (419), `TInterfaceList.GetEnumerator` (422), `#rtl.system.IEnumerator` (1591)

4.58.2 Method overview

Page	Method	Description
425	<code>Create</code>	Initialize a new instance of <code>TInterfaceListEnumerator</code> .
425	<code>GetCurrent</code>	Return the current pointer in the list.
426	<code>MoveNext</code>	Move the position of the enumerator to the next position in the children of the component.

4.58.3 Property overview

Page	Properties	Access	Description
426	<code>Current</code>	<code>r</code>	Current pointer in the list.

4.58.4 TInterfaceListEnumerator.Create

Synopsis: Initialize a new instance of `TInterfaceListEnumerator`.

Declaration: `constructor Create (AList: TInterfaceList)`

Visibility: `public`

Description: `Create` initializes a new instance of `TInterfaceListEnumerator` and keeps a reference to the component `AComponent` that will be enumerated.

See also: `TInterfaceList` (419)

4.58.5 TInterfaceListEnumerator.GetCurrent

Synopsis: Return the current pointer in the list.

Declaration: `function GetCurrent : IUnknown`

Visibility: `public`

Description: `GetCurrent` returns the current interface in the `TInterfaceList` (419) list.

Errors: No checking is done on the validity of the current position.

See also: `MoveNext` (426), `TInterfaceList.Components` (419)

4.58.6 TInterfaceListEnumerator.MoveNext

Synopsis: Move the position of the enumerator to the next position in the children of the component.

Declaration: `function MoveNext : Boolean`

Visibility: `public`

Description: `MoveNext` puts the pointer on the next interface in the list, and returns `True` if this succeeded, or `False` if the pointer is past the last interface in the list.

Errors: Note that if `False` is returned, calling `GetCurrent` will result in an exception.

See also: `GetCurrent` ([425](#))

4.58.7 TInterfaceListEnumerator.Current

Synopsis: Current pointer in the list.

Declaration: `Property Current : IUnknown`

Visibility: `public`

Access: `Read`

Description: `Current` redefines `GetCurrent` ([425](#)) as a property.

See also: `GetCurrent` ([425](#))

4.59 TList

4.59.1 Description

`TList` is a class that can be used to manage collections of pointers. It introduces methods and properties to store the pointers, search in the list of pointers, sort them. It manages its memory by itself, no intervention for that is needed. It has an event notification mechanism which allows to notify of list changes. This slows down some of `TList` mechanisms, and if no notification is used, `TFPList` ([408](#)) may be used instead.

To manage collections of strings, it is better to use a `TStrings` ([486](#)) descendant such as `TStringList` ([481](#)). To manage general objects, a `TCollection` ([376](#)) class exists, from which a descendant can be made to manage collections of various kinds.

See also: `TStrings` ([486](#)), `TCollection` ([376](#))

4.59.2 Interfaces overview

Page	Interfaces	Description
323	<code>IFPObserved</code>	Interface implemented by an object that can be observed.

4.59.3 Method overview

Page	Method	Description
429	Add	Adds a new pointer to the list.
429	AddList	Add all pointers from another list.
432	Assign	Copy the contents of other lists.
429	Clear	Clears the pointer list.
431	Contains	
427	Create	Class to manage collections of pointers.
429	Delete	Removes a pointer from the list.
427	Destroy	Destroys the list and releases the memory used to store the list elements.
430	Error	Raises an <code>EListError</code> (284) exception.
430	Exchange	Exchanges two pointers in the list.
430	Expand	Increases the capacity of the list if needed.
430	Extract	Remove the first occurrence of a pointer from the list.
431	First	Returns the first non-nil pointer in the list.
428	FPOAttachObserver	Add an observer to the list of observers.
428	FPODetachObserver	Remove an observer from the list of observers.
428	FPONotifyObservers	Notify observers of changes in the list.
431	GetEnumerator	Create an <code>IEnumerator</code> instance.
431	IndexOf	Returns the index of a given pointer.
431	Insert	Inserts a new pointer in the list at a given position.
432	Last	Returns the last non-nil pointer in the list.
432	Move	Moves a pointer from one position in the list to another.
433	Pack	Removes <code>Nil</code> pointers from the list and frees unused memory.
432	Remove	Removes a value from the list.
433	Sort	Sorts the pointers in the list.

4.59.4 Property overview

Page	Properties	Access	Description
433	Capacity	rw	Current capacity (i.e. number of pointers that can be stored) of the list.
434	Count	rw	Current number of pointers in the list.
434	Items	rw	Provides access to the pointers in the list.
434	List	r	Memory array where pointers are stored.

4.59.5 TList.Create

Synopsis: Class to manage collections of pointers.

Declaration: `constructor Create`

Visibility: `public`

Description: `TList.Create` creates a new instance of `TList`. It clears the list and prepares it for use.

See also: `TList` ([426](#)), `TList.Destroy` ([427](#))

4.59.6 TList.Destroy

Synopsis: Destroys the list and releases the memory used to store the list elements.

Declaration: `destructor Destroy; Override`

Visibility: public

Description: `Destroy` destroys the list and releases the memory used to store the list elements. The elements themselves are in no way touched, i.e. any memory they point to must be explicitly released before calling the destructor.

4.59.7 TList.FPOAttachObserver

Synopsis: Add an observer to the list of observers.

Declaration: `procedure FPOAttachObserver(AObserver: TObject)`

Visibility: public

Description: `FPOAttachObserver` is part of the implementation of the `IFPObserved` (323) interface in `TList`. It adds a new observer to the list of observers. Calling this multiple times will add the observed object multiple times to the list.

Errors: An `EObserver` exception may be raised if `AObject` does not implement the `IFPObserver` (325) interface.

See also: `IFPObserver` (325), `IFPObserved.FPOAttachObserver` (324), `IFPObserved` (323)

4.59.8 TList.FPODetachObserver

Synopsis: Remove an observer from the list of observers.

Declaration: `procedure FPODetachObserver(AObserver: TObject)`

Visibility: public

Description: `FPODetachObserver` is part of the implementation of the `IFPObserved` (323) interface in `TList`. It removes the first found instance of the observer from the list of observers.

See also: `IFPObserved` (323), `IFPObserved.FPODetachObserver` (324), `IFPObserver` (325)

4.59.9 TList.FPONotifyObservers

Synopsis: Notify observers of changes in the list.

Declaration: `procedure FPONotifyObservers(ASender: TObject;
AOperation: TFPObservedOperation;
Data: Pointer)`

Visibility: public

Description: `FPONotifyObservers` is called to notify observers of changes in the list. The following notifications are sent:

ooAddItem when a pointer is added. `Data` is the pointer that is added.

ooDeleteItem when a pointer is deleted or extracted. `Data` is the pointer that is deleted or extracted.

ooChange called when 2 pointers are exchanged.

ooFree Called when the list is freed.

See also: `FPODetachObserver` (281), `FPOAttachObserver` (281), `Add` (281), `Exchange` (281), `Delete` (281), `Extract` (281)

4.59.10 TList.AddList

Synopsis: Add all pointers from another list.

Declaration: `procedure AddList (AList: TList)`

Visibility: `public`

Description: `AddList` adds all pointers from `AList` to the list. If a pointer is already present, it is added a second time.

See also: `TList.Assign` (432), `TFPList.AddList` (409)

4.59.11 TList.Add

Synopsis: Adds a new pointer to the list.

Declaration: `function Add (Item: Pointer) : Integer`

Visibility: `public`

Description: `Add` adds a new pointer to the list after the last pointer (i.e. at position `Count`, thus increasing the item count with 1. If the list is at full capacity, the capacity of the list is expanded, using the `Grow` (426) method.

To insert a pointer at a certain position in the list, use the `Insert` (431) method instead.

See also: `Delete` (429), `Grow` (426), `Insert` (431)

4.59.12 TList.Clear

Synopsis: Clears the pointer list.

Declaration: `procedure Clear; Virtual`

Visibility: `public`

Description: `Clear` removes all pointers from the list, and sets the capacity to 0, thus freeing any memory allocated to maintain the list.

See also: `Destroy` (427)

4.59.13 TList.Delete

Synopsis: Removes a pointer from the list.

Declaration: `procedure Delete (Index: Integer)`

Visibility: `public`

Description: `Delete` removes the pointer at position `Index` from the list, shifting all following pointers one position up (or to the left).

The memory the pointer is pointing to is *not* deallocated.

4.59.14 TList.Error

Synopsis: Raises an `EListError` (284) exception.

Declaration: `class procedure Error(const Msg: string; Data: PtrInt); Virtual`

Visibility: `public`

Description: `Error` raises an `EListError` (284) exception, with a message formatted with `Msg` and `Data`.

4.59.15 TList.Exchange

Synopsis: Exchanges two pointers in the list.

Declaration: `procedure Exchange(Index1: Integer; Index2: Integer)`

Visibility: `public`

Description: `Exchange` exchanges the pointers at positions `Index1` and `Index2`. Both pointers must be within the current range of the list, or an `EListError` (284) exception will be raised.

4.59.16 TList.Expand

Synopsis: Increases the capacity of the list if needed.

Declaration: `function Expand : TList`

Visibility: `public`

Description: `Expand` increases the capacity of the list if the current element count matches the current list capacity.

The capacity is increased according to the following algorithm:

- 1.If the capacity is less than 3, the capacity is increased with 4.
- 2.If the capacity is larger than 3 and less than 8, the capacity is increased with 8.
- 3.If the capacity is larger than 8, the capacity is increased with 16.

The return value is `Self`.

See also: `Capacity` (433)

4.59.17 TList.Extract

Synopsis: Remove the first occurrence of a pointer from the list.

Declaration: `function Extract(item: Pointer) : Pointer`

Visibility: `public`

Description: `Extract` searched for an occurrence of `item`, and if a match is found, the match is deleted from the list. If no match is found, nothing is deleted. If `Item` was found, the result is `Item`. If `Item` was not found, the result is `Nil`. A `lnExtracted` notification event is triggered if an element is extracted from the list.

See also: `TList.Delete` (429), `TList.IndexOf` (431), `TList.Remove` (432)

4.59.18 TList.First

Synopsis: Returns the first non-nil pointer in the list.

Declaration: `function First : Pointer`

Visibility: public

Description: `First` returns the value of the first non-nil pointer in the list.

If there are no pointers in the list or all pointers equal `Nil`, then `Nil` is returned.

See also: `Last` ([432](#))

4.59.19 TList.GetEnumerator

Synopsis: Create an `IEnumerator` instance.

Declaration: `function GetEnumerator : TListEnumerator`

Visibility: public

Description: `GetEnumerator` is the implementation of the `IEnumerable` ([1591](#)) interface for `TList`. It creates a `TListEnumerator` ([434](#)) instance and returns it's `IEnumerator` ([1591](#)) interface.

See also: `TListEnumerator` ([434](#)), `IEnumerator` ([1591](#)), `IEnumerable` ([1591](#))

4.59.20 TList.Contains

Declaration: `function Contains(Item: Pointer) : Boolean`

Visibility: public

4.59.21 TList.IndexOf

Synopsis: Returns the index of a given pointer.

Declaration: `function IndexOf(Item: Pointer) : Integer`

Visibility: public

Description: `IndexOf` searches for the pointer `Item` in the list of pointers, and returns the index of the pointer, if found.

If no pointer with the value `Item` was found, -1 is returned.

4.59.22 TList.Insert

Synopsis: Inserts a new pointer in the list at a given position.

Declaration: `procedure Insert(Index: Integer; Item: Pointer)`

Visibility: public

Description: `Insert` inserts pointer `Item` at position `Index` in the list. All pointers starting from `Index` are shifted to the right.

If `Index` is not a valid position, then a `EListError` ([284](#)) exception is raised.

See also: `Add` ([429](#)), `Delete` ([429](#))

4.59.23 TList.Last

Synopsis: Returns the last non-nil pointer in the list.

Declaration: `function Last : Pointer`

Visibility: `public`

Description: `Last` returns the value of the last non-nil pointer in the list.

If there are no pointers in the list or all pointers equal `Nil`, then `Nil` is returned.

See also: `First` ([431](#))

4.59.24 TList.Move

Synopsis: Moves a pointer from one position in the list to another.

Declaration: `procedure Move (CurIndex: Integer; NewIndex: Integer)`

Visibility: `public`

Description: `Move` moves the pointer at position `CurIndex` to position `NewIndex`. This is done by storing the value at position `CurIndex`, deleting the pointer at position `CurIndex`, and reinserting the value at position `NewIndex`.

If `CurIndex` or `Newindex` are not inside the valid range of indices, an `EListError` ([284](#)) exception is raised.

See also: `Exchange` ([430](#))

4.59.25 TList.Assign

Synopsis: Copy the contents of other lists.

Declaration: `procedure Assign (ListA: TList; AOperator: TListAssignOp; ListB: TList)`

Visibility: `public`

Description: `Assign` can be used to merge or assign lists. It is an extended version of the usual `TPersistent.Assign` mechanism. The arguments `ListA` and `ListB` are used as sources of pointers to add or remove elements from the current list, depending on the operation `AOperation`. The available operations are documented in the `TListAssignOp` ([291](#)) type.

See also: `TList.Clear` ([429](#))

4.59.26 TList.Remove

Synopsis: Removes a value from the list.

Declaration: `function Remove (Item: Pointer) : Integer`

Visibility: `public`

Description: `Remove` searches `Item` in the list, and, if it finds it, deletes the item from the list. Only the first occurrence of `Item` is removed.

See also: `Delete` ([429](#)), `IndexOf` ([431](#)), `Insert` ([431](#))

4.59.27 TList.Pack

Synopsis: Removes Nil pointers from the list and frees unused memory.

Declaration: `procedure Pack`

Visibility: `public`

Description: `Pack` removes all `nil` pointers from the list. The capacity of the list is then set to the number of pointers in the list. This method can be used to free unused memory if the list has grown to very large sizes and has a lot of unneeded `nil` pointers in it.

See also: `TList.Clear` ([429](#))

4.59.28 TList.Sort

Synopsis: Sorts the pointers in the list.

Declaration: `procedure Sort (Compare: TListSortCompare)`

Visibility: `public`

Description: `Sort` sorts the pointers in the list. Two pointers are compared by passing them to the `Compare` function. The result of this function determines how the pointers will be sorted:

- If the result of this function is negative, the first pointer is assumed to be 'less' than the second and will be moved before the second in the list.
- If the function result is positive, the first pointer is assumed to be 'greater than' the second and will be moved after the second in the list.
- If the function result is zero, the pointers are assumed to be 'equal' and no moving will take place.

The sort is done using a quicksort algorithm.

4.59.29 TList.Capacity

Synopsis: Current capacity (i.e. number of pointers that can be stored) of the list.

Declaration: `Property Capacity : Integer`

Visibility: `public`

Access: `Read, Write`

Description: `Capacity` contains the number of pointers the list can store before it starts to grow.

If a new pointer is added to the list using `add` ([429](#)) or `insert` ([431](#)), and there is not enough memory to store the new pointer, then the list will try to allocate more memory to store the new pointer. Since this is a time consuming operation, it is important that this operation be performed as little as possible. If it is known how many pointers there will be before filling the list, it is a good idea to set the capacity first before filling. This ensures that the list doesn't need to grow, and will speed up filling the list.

See also: `SetCapacity` ([426](#)), `Count` ([434](#))

4.59.30 TList.Count

Synopsis: Current number of pointers in the list.

Declaration: `Property Count : Integer`

Visibility: `public`

Access: `Read,Write`

Description: `Count` is the current number of (possibly `Nil`) pointers in the list. Since the list is zero-based, the index of the largest pointer is `Count-1`.

4.59.31 TList.Items

Synopsis: Provides access to the pointers in the list.

Declaration: `Property Items[Index: Integer]: Pointer; default`

Visibility: `public`

Access: `Read,Write`

Description: `Items` is used to access the pointers in the list. It is the default property of the `TList` class, so it can be omitted.

The list is zero-based, so `Index` must be in the range 0 to `Count-1`.

4.59.32 TList.List

Synopsis: Memory array where pointers are stored.

Declaration: `Property List : PPointerList`

Visibility: `public`

Access: `Read`

Description: `List` points to the memory space where the pointers are stored. This can be used to quickly copy the list of pointers to another location.

4.60 TListEnumerator

4.60.1 Description

`TListEnumerator` implements the `#rtl.system.IEnumerator` (1591) interface for the `TList` (426) class, so the `TList` class can be used in a `for ... in` loop. It is returned by the `TList.GetEnumerator` (431) method of `TList`.

See also: `TList` (426), `TList.GetEnumerator` (431), `#rtl.system.IEnumerator` (1591)

4.60.2 Method overview

Page	Method	Description
435	<code>Create</code>	Initialize a new instance of <code>TListEnumerator</code> .
435	<code>GetCurrent</code>	Return the current pointer in the list.
435	<code>MoveNext</code>	Move the position of the enumerator to the next position in the list.

4.60.3 Property overview

Page	Properties	Access	Description
435	Current	r	Current pointer in the list.

4.60.4 TListEnumerator.Create

Synopsis: Initialize a new instance of `TListEnumerator`.

Declaration: `constructor Create (AList: TList)`

Visibility: `public`

Description: `Create` initializes a new instance of `TListEnumerator` and keeps a reference to the list `AList` that will be enumerated.

See also: `TList` ([426](#))

4.60.5 TListEnumerator.GetCurrent

Synopsis: Return the current pointer in the list.

Declaration: `function GetCurrent : Pointer`

Visibility: `public`

Description: `GetCurrent` returns the current pointer in the enumerator.

Errors: No checking is done on the validity of the current position.

See also: `MoveNext` ([435](#))

4.60.6 TListEnumerator.MoveNext

Synopsis: Move the position of the enumerator to the next position in the list.

Declaration: `function MoveNext : Boolean`

Visibility: `public`

Description: `MoveNext` puts the pointer on the next item in the list, and returns `True` if this succeeded, or `False` if the pointer is past the last element in the list.

Errors: Note that if `False` is returned, calling `GetCurrent` will result in an exception.

See also: `GetCurrent` ([435](#))

4.60.7 TListEnumerator.Current

Synopsis: Current pointer in the list.

Declaration: `Property Current : Pointer`

Visibility: `public`

Access: `Read`

Description: `Current` redefines `GetCurrent` ([435](#)) as a property.

See also: `GetCurrent` ([435](#))

4.61 TMemoryStream

4.61.1 Description

`TMemoryStream` is a `TStream` (466) descendant that stores its data in memory. It descends directly from `TCustomMemoryStream` (398) and implements the necessary to allocate and de-allocate memory directly from the heap. It implements the `Write` (437) method which is missing in `TCustomMemoryStream`.

`TMemoryStream` also introduces methods to load the contents of another stream or a file into the memory stream.

It is not necessary to do any memory management manually, as the stream will allocate or de-allocate memory as needed. When the stream is freed, all allocated memory will be freed as well.

See also: `TCustomMemoryStream` (398), `TStream` (466)

4.61.2 Method overview

Page	Method	Description
436	<code>Clear</code>	Zeroes the position, capacity and size of the stream.
436	<code>Destroy</code>	Frees any allocated memory and destroys the memory stream.
437	<code>LoadFromFile</code>	Loads the contents of a file into memory.
436	<code>LoadFromStream</code>	Loads the contents of a stream into memory.
437	<code>SetSize</code>	Sets the size for the memory stream.
437	<code>Write</code>	Writes data to the stream's memory.

4.61.3 TMemoryStream.Destroy

Synopsis: Frees any allocated memory and destroys the memory stream.

Declaration: `destructor Destroy; Override`

Visibility: `public`

Description: `Destroy` clears the memory stream, thus in effect freeing any memory allocated for it, and then frees the memory stream.

4.61.4 TMemoryStream.Clear

Synopsis: Zeroes the position, capacity and size of the stream.

Declaration: `procedure Clear`

Visibility: `public`

Description: `Clear` sets the position and size to 0, and sets the capacity of the stream to 0, thus freeing all memory allocated for the stream.

See also: `TStream.Size` (476), `TStream.Position` (475), `TCustomMemoryStream.Memory` (400)

4.61.5 TMemoryStream.LoadFromStream

Synopsis: Loads the contents of a stream into memory.

Declaration: `procedure LoadFromStream(Stream: TStream)`

Visibility: public

Description: `LoadFromStream` loads the contents of `Stream` into the memorybuffer of the stream. Any previous contents of the memory stream are overwritten. Memory is allocated as needed.

Remark The `LoadFromStream` uses the `Size` (476) property of `Stream` to determine how much memory must be allocated. Some streams do not allow the stream size to be determined, so care must be taken when using this method.

This method will work much faster than the use of the `TStream.CopyFrom` (470) method:

```
Seek(0, soFromBeginning);
CopyFrom(Stream, Stream.Size);
```

because the `CopyFrom` method copies the contents in blocks, while `LoadFromStream` reads the contents of the stream as one big block.

Errors: If an error occurs when reading from the stream, an `EStreamError` (321) may occur.

See also: `TStream.CopyFrom` (470), `TMemoryStream.LoadFromFile` (437)

4.61.6 TMemoryStream.LoadFromFile

Synopsis: Loads the contents of a file into memory.

Declaration: `procedure LoadFromFile(const FileName: string)`

Visibility: public

Description: `LoadFromFile` loads the contents of the file with name `FileName` into the memory stream. The current contents of the memory stream is replaced by the contents of the file. Memory is allocated as needed.

The `LoadFromFile` method simply creates a filestream and then calls the `TMemoryStream.LoadFromStream` (436) method.

See also: `TMemoryStream.LoadFromStream` (436)

4.61.7 TMemoryStream.SetSize

Synopsis: Sets the size for the memory stream.

Declaration: `procedure SetSize(const NewSize: Int64); Override`

Visibility: public

Description: `SetSize` sets the size of the memory stream to `NewSize`. This will set the capacity of the stream to `NewSize` and correct the current position in the stream when needed.

See also: `TStream.Position` (475), `TStream.Size` (476)

4.61.8 TMemoryStream.Write

Synopsis: Writes data to the stream's memory.

Declaration: `function Write(const Buffer; Count: LongInt) : LongInt; Override`

Visibility: public

Description: `Write` writes `Count` bytes from `Buffer` to the stream's memory, starting at the current position in the stream. If more memory is needed than currently allocated, more memory will be allocated. Any contents in the memory stream at the current position will be overwritten. The function returns the number of bytes actually written (which should under normal circumstances always equal `Count`).

This method overrides the `TStream.Write` (468) method.

Errors: If no more memory could be allocated, then an exception will be raised.

See also: `TCustomMemoryStream.Read` (398)

4.62 TOwnedCollection

4.62.1 Description

`TOwnedCollection` automatically maintains owner information, so it can be displayed in an IDE. Collections that should be displayed in an IDE should descend from `TOwnedCollection` or must implement a `GetOwner` function.

See also: `TCollection` (376)

4.62.2 Method overview

Page	Method	Description
438	<code>Create</code>	Create a new <code>TOwnerCollection</code> instance.

4.62.3 TOwnedCollection.Create

Synopsis: Create a new `TOwnerCollection` instance.

Declaration: `constructor Create(AOwner: TPersistent;
 AItemClass: TCollectionItemClass)`

Visibility: `public`

Description: `Create` creates a new instance of `TOwnedCollection` and stores the `AOwner` references. It will the value returned in the `TCollection.Owner` (377) property of the collection. The `ItemClass` class reference is passed on to the inherited constructor, and will be used to create new instances in the `Insert` (380) and `Add` (377) methods.

See also: `TCollection.Create` (376), `TCollection.Owner` (377)

4.63 TOwnerStream

4.63.1 Description

`TOwnerStream` can be used when creating stream chains such as when using encryption and compression streams. It keeps a reference to the source stream and will automatically free the source stream when ready (if the `SourceOwner` (440) property is set to `True`).

See also: `TStream` (466), `TOwnerStream.Source` (439), `TOwnerStream.SourceOwner` (440)

4.63.2 Method overview

Page	Method	Description
439	Create	Create a new instance of <code>TOwnerStream</code> .
439	Destroy	Destroys the <code>TOwnerStream</code> instance and the source stream.

4.63.3 Property overview

Page	Properties	Access	Description
439	Source	r	Reference to the source stream.
440	SourceOwner	rw	Indicates whether the ownerstream owns it's source.

4.63.4 TOwnerStream.Create

Synopsis: Create a new instance of `TOwnerStream`.

Declaration: `constructor Create (ASource: TStream)`

Visibility: `public`

Description: `Create` instantiates a new instance of `TOwnerStream` and stores the reference to `ASource`. If `SourceOwner` is `True`, the source stream will also be freed when the instance is destroyed.

See also: `TOwnerStream.Destroy` ([439](#)), `TOwnerStream.Source` ([439](#)), `TOwnerStream.SourceOwner` ([440](#))

4.63.5 TOwnerStream.Destroy

Synopsis: Destroys the `TOwnerStream` instance and the source stream.

Declaration: `destructor Destroy; Override`

Visibility: `public`

Description: `Destroy` frees the source stream if the `SourceOwner` property is `True`.

See also: `TOwnerStream.Create` ([439](#)), `TOwnerStream.Source` ([439](#)), `TOwnerStream.SourceOwner` ([440](#))

4.63.6 TOwnerStream.Source

Synopsis: Reference to the source stream.

Declaration: `Property Source : TStream`

Visibility: `public`

Access: `Read`

Description: `Source` is the source stream. It should be used by descendant streams to access the source stream to read from or write to.

Do not free the `Source` reference directly if `SourceOwner` is `True`. In that case the owner stream instance will free the source stream itself.

See also: `TOwnerStream.Create` ([439](#))

4.63.7 TOwnerStream.SourceOwner

Synopsis: Indicates whether the ownerstream owns it's source.

Declaration: `Property SourceOwner : Boolean`

Visibility: `public`

Access: `Read, Write`

Description: `SourceOwner` indicates whether the `TOwnerStream` owns it's `Source` stream or not. If this property is `True` then the `Source` stream is freed when the `TOwnerStream` instance is freed.

See also: `TOwnerStream.Source` ([439](#)), `TOwnerStream.Destroy` ([439](#))

4.64 TParser

4.64.1 Description

This class breaks a stream of text data in tokens. Its primary use is to help reading the contents of a form file (usually a file with `dfm`, `xfm` or `lfm` extension), and for this reason it isn't suitable to be used as a general parser.

The parser is always positioned on a certain token, whose type is stored in the `Token` ([446](#)) property. Various methods are provided to obtain the token value in the desired format.

To advance to the next token, invoke `NextToken` ([443](#)) method.

See also: `TParser.Token` ([446](#)), `TParser.NextToken` ([443](#))

4.64.2 Method overview

Page	Method	Description
441	<code>CheckToken</code>	Checks whether the token if of the given type.
441	<code>CheckTokenSymbol</code>	Checks whether the token equals the given symbol.
441	<code>Create</code>	Creates a new parser instance.
441	<code>Destroy</code>	Destroys the parser instance.
442	<code>Error</code>	Raises an <code>EParserError</code> (321) exception with the given message.
442	<code>ErrorFmt</code>	Raises an <code>EParserError</code> (321) exception and formats the message.
442	<code>ErrorStr</code>	Raises an <code>EParserError</code> (321) exception with the given message.
442	<code>HexToBinary</code>	Writes hexadecimal data to a stream.
443	<code>NextToken</code>	Reads the next token and returns its type.
443	<code>SourcePos</code>	Returns the current position in the stream.
443	<code>TokenComponentIdent</code>	Returns the path of a subcomponent starting from the current token.
444	<code>TokenFloat</code>	Returns the current token as a float.
444	<code>TokenInt</code>	Returns the current token as an integer.
444	<code>TokenString</code>	Returns the current token as a string.
445	<code>TokenSymbolIs</code>	Returns <code>True</code> if the token equals the given symbol.
445	<code>TokenWideString</code>	Returns the current token as a widestring.

4.64.3 Property overview

Page	Properties	Access	Description
445	FloatType	r	The type of a float token.
446	SourceLine	r	Current source line number.
446	Token	r	The type of the current token.

4.64.4 TParser.Create

Synopsis: Creates a new parser instance.

Declaration: `constructor Create(Stream: TStream)`

Visibility: `public`

Description: `Create` creates a new `TParser` instance, using `Stream` as the stream to read data from, and reads the first token from the stream.

Errors: If an error occurs while parsing the first token, an `EParserError` ([321](#)) exception is raised.

See also: `TParser.NextToken` ([443](#)), `TParser.Token` ([446](#))

4.64.5 TParser.Destroy

Synopsis: Destroys the parser instance.

Declaration: `destructor Destroy; Override`

Visibility: `public`

Description: Destroys the parser instance.

Errors: None.

4.64.6 TParser.CheckToken

Synopsis: Checks whether the token is of the given type.

Declaration: `procedure CheckToken(T: Char)`

Visibility: `public`

Description: Checks whether the token is of the given type.

Errors: If current token isn't of type `T`, an `EParserError` ([321](#)) exception is raised.

See also: `TParser.Token` ([446](#))

4.64.7 TParser.CheckTokenSymbol

Synopsis: Checks whether the token equals the given symbol.

Declaration: `procedure CheckTokenSymbol(const S: string)`

Visibility: `public`

Description: `CheckTokenSymbol` performs a case-insensitive comparison of current token value with `S`.

Current token must be of type `toSymbol` ([283](#)), otherwise an `EParserError` ([321](#)) exception is raised.

Errors: If the comparison fails, or current token isn't a symbol, an `EParseError` (321) exception is raised.

See also: `TParser.TokenSymbolIs` (445), `toSymbol` (283)

4.64.8 `TParser.Error`

Synopsis: Raises an `EParseError` (321) exception with the given message.

Declaration: `procedure Error(const Ident: string)`

Visibility: public

Description: Raises an `EParseError` (321) exception with the given message.

4.64.9 `TParser.ErrorFmt`

Synopsis: Raises an `EParseError` (321) exception and formats the message.

Declaration: `procedure ErrorFmt(const Ident: string; const Args: Array of const)`

Visibility: public

Description: Raises an `EParseError` (321) exception and formats the message.

4.64.10 `TParser.ErrorStr`

Synopsis: Raises an `EParseError` (321) exception with the given message.

Declaration: `procedure ErrorStr(const Message: string)`

Visibility: public

Description: Raises an `EParseError` (321) exception with the given message.

4.64.11 `TParser.HexToBinary`

Synopsis: Writes hexadecimal data to a stream.

Declaration: `procedure HexToBinary(Stream: TStream)`

Visibility: public

Description: `HexToBinary` reads a sequence of hexadecimal characters from the input stream and converts them to a sequence of bytes which is written to `Stream`. Each byte is represented by two contiguous hexadecimal characters.

Whitespace is allowed between hexadecimal characters if it doesn't appear between two characters that form the same byte.

`HexToBinary` stops when the first non-hexadecimal and non-whitespace character is found, or the end of the input stream is reached.

Remark This method begins reading after the current token: that is, current token, even if it's a valid hexadecimal value, isn't included.

Errors: If a single hexadecimal character is found, an `EParseError` (321) exception is raised.

4.64.12 TParser.NextToken

Synopsis: Reads the next token and returns its type.

Declaration: `function NextToken : Char`

Visibility: public

Description: `NextToken` parses the next token in the stream and returns its type. The type of the token can also be retrieved later reading `Token` (446) property.

If the end of the stream is reached, `toEOF` (283) is returned.

For details about token types, see `TParser.Token` (446)

Errors: If an error occurs while parsing the token, an `EParserError` (321) exception is raised.

See also: `TParser.Token` (446)

4.64.13 TParser.SourcePos

Synopsis: Returns the current position in the stream.

Declaration: `function SourcePos : LongInt`

Visibility: public

Description: This is not the character position relative to the current source line, but the byte offset from the beginning of the stream.

Errors: None.

See also: `TParser.SourceLine` (446)

4.64.14 TParser.TokenComponentIdent

Synopsis: Returns the path of a subcomponent starting from the current token.

Declaration: `function TokenComponentIdent : string`

Visibility: public

Description: If current token is `toSymbol` (283), `TokenComponentIdent` tries to find subcomponent names separated by a dot (.). The returned string is the longest subcomponent path found. If there are no subcomponents, current symbol is returned.

Remark After this method has been called, subsequent calls to `TokenString` (444) or `TokenWideString` (445) return the same value returned by `TokenComponentIdent`.

Example

If source stream contains `a.b.c` and `TParser` is positioned on the first token (`a`), this method returns `a.b.c`.

Errors: If `Token` (446) isn't `toSymbol` (283), or no valid symbol is found after a dot, an `EParserError` (321) exception is raised.

See also: `TParser.NextToken` (443), `TParser.Token` (446), `TParser.TokenString` (444), `TParser.TokenWideString` (445), `toSymbol` (283)

4.64.15 TParser.TokenFloat

Synopsis: Returns the current token as a float.

Declaration: `function TokenFloat : Extended`

Visibility: `public`

Description: If current token type is `toFloat` (283), this method returns the token value as a float.

To specify a negative number, no space must exist between unary minus and number.

Floating point numbers can be postfixed with a character that specifies the floating point type. See `FloatType` (445) for further information.

Remark In the input stream the decimal separator, if present, must be a dot (.).

Errors: If `Token` (446) isn't `toFloat` (283), an `EParserError` (321) exception is raised.

See also: `TParser.FloatType` (445), `TParser.NextToken` (443), `TParser.Token` (446), `toFloat` (283)

4.64.16 TParser.TokenInt

Synopsis: Returns the current token as an integer.

Declaration: `function TokenInt : Int64`

Visibility: `public`

Description: If current token type is `toInteger` (283), this method returns the token value as an integer.

In the input stream an integer can be an hexadecimal (prefixed by ' \$ ' character) or decimal number. Decimal numbers can be prefixed by an unary minus: if this is the case, no space must exist between minus and number.

Errors: If `Token` (446) isn't `toInteger` (283), an `EConvertError` (281) exception is raised.

See also: `TParser.NextToken` (443), `TParser.Token` (446), `toInteger` (283)

4.64.17 TParser.TokenString

Synopsis: Returns the current token as a string.

Declaration: `function TokenString : string`

Visibility: `public`

Description: If current token type is `toString` (283) or `toWString` (283), this method returns the contents of the string. That is, enclosing quotes are removed, embedded quotes are unescaped and control strings are converted to the appropriate sequence of characters.

If current token type isn't a string, a string containing the token representation in the input stream is returned, without any conversion: hexadecimal integers are returned with the leading \$, and floating point suffixes like `s`, `c` or `d` are kept. For tokens whose type isn't a special type, return value of `TokenString` equals `Token` (446).

Remark If `Token` (446) is `toWString` (283), `TokenWideString` (445) should be used instead.

Errors: None.

See also: `TParser.NextToken` (443), `TParser.TokenWideString` (445), `TParser.Token` (446), `toString` (283), `toWString` (283)

4.64.18 TParser.TokenWideString

Synopsis: Returns the current token as a widestring.

Declaration: `function TokenWideString : WideString`

Visibility: public

Description: If current token type is `toWString` (283), this method returns the contents of the string. That is, enclosing quotes are removed, embedded quotes are unescaped and control strings are converted to the appropriate sequence of characters.

If current token isn't a widestring, `TokenWideString` behaviour is the same as `TokenString` (444).

Errors: None.

See also: `TParser.NextToken` (443), `TokenString` (444), `TParser.Token` (446), `toWString` (283)

4.64.19 TParser.TokenSymbols

Synopsis: Returns `True` if the token equals the given symbol.

Declaration: `function TokenSymbolIs(const S: string) : Boolean`

Visibility: public

Description: `TokenSymbolIs` performs a case-insensitive comparison of current token value with `S`.

If current token isn't of type `toSymbol` (283), or comparison fails, `False` is returned.

Errors: None.

See also: `TParser.CheckTokenSymbol` (441), `TParser.Token` (446)

4.64.20 TParser.FloatType

Synopsis: The type of a float token.

Declaration: `Property FloatType : Char`

Visibility: public

Access: Read

Description: Floating point numbers can be postfixed with a character specifying the type of floating point value.

When specified, this property holds the character postfixed to the number.

It can be one of the following values:

Table 4.28:

s or S	Value is a single.
c or C	Value is a currency.
d or D	Value is a date.

If `Token` (446) isn't `toFloat` (283) or one of the above characters wasn't specified, `FloatType` is the null character (zero).

See also: `TParser.NextToken` (443), `TParser.Token` (446), `TParser.TokenFloat` (444), `toFloat` (283)

4.64.21 TParser.SourceLine

Synopsis: Current source line number.

Declaration: `Property SourceLine : Integer`

Visibility: `public`

Access: `Read`

Description: Current source line number.

See also: `TParser.SourcePos` (443)

4.64.22 TParser.Token

Synopsis: The type of the current token.

Declaration: `Property Token : Char`

Visibility: `public`

Access: `Read`

Description: This property holds the type of the current token. When `Token` isn't one of the special token types (whose value can be retrieved with specific methods) it is the character representing the current token.

Special token types:

Table 4.29:

<code>toEOF</code> (283)	Value returned by <code>TParser.Token</code> (446) when the end of the input stream was reached.
<code>toSymbol</code> (283)	Value returned by <code>TParser.Token</code> (446) when a symbol was found in the input stream.
<code>toString</code> (283)	Value returned by <code>TParser.Token</code> (446) when a string was found in the input stream.
<code>toInteger</code> (283)	Value returned by <code>TParser.Token</code> (446) when an integer was found in the input stream.
<code>toFloat</code> (283)	Value returned by <code>TParser.Token</code> (446) when a floating point value was found in the input stream.
<code>toWString</code> (283)	Value returned by <code>TParser.Token</code> (446) when a widestring was found in the input stream.

To advance to the next token, use `NextToken` (443) method.

See also: `TParser.CheckToken` (441), `TParser.NextToken` (443), `TParser.TokenComponentIdent` (443), `TParser.TokenFloat` (444), `TParser.TokenInt` (444), `TParser.TokenString` (444), `TParser.TokenWideString` (445)

4.65 TPersistent

4.65.1 Description

`TPersistent` is the basic class for the streaming system. Since it is compiled in the `{ $M+ }` state, the compiler generates RTTI (Run-Time Type Information) for it and all classes that descend from it. This information can be used to stream all properties of classes.

It also introduces functionality to assign the contents of 2 classes to each other.

`TPersistent` implements the `IFPObserved` (323) interface for the benefit of descendant classes, but does not call `IFPObserved.FPONotifyObservers` (324). Descendants such as `TStrings` (486) and `TCollection` (376) and `TCollectionItem` (383) do use it.

See also: `TComponent` (386), `IFPObserved` (323), `TStrings` (486), `TCollection` (376), `TCollectionItem` (383)

4.65.2 Interfaces overview

Page	Interfaces	Description
323	IFPObserved	Interface implemented by an object that can be observed.

4.65.3 Method overview

Page	Method	Description
447	Assign	Assign the contents of one class to another.
447	Destroy	Destroys the <code>TPersistent</code> instance.
448	FPOAttachObserver	Add an observer to the list of observers.
448	FPODetachObserver	Remove an observer from the list of observers.
448	FPONotifyObservers	Notify observers of changes.
448	GetNamePath	Returns a string that can be used to identify the class instance.

4.65.4 `TPersistent.Destroy`

Synopsis: Destroys the `TPersistent` instance.

Declaration: `destructor Destroy; Override`

Visibility: `public`

Description: `Destroy` disposes of the persistent object. This method should never be called directly. Instead the `Free` method should be used.

4.65.5 `TPersistent.Assign`

Synopsis: Assign the contents of one class to another.

Declaration: `procedure Assign(Source: TPersistent); Virtual`

Visibility: `public`

Description: `Assign` copies the contents of `Source` to `Self`, if the classes of the destination and source classes are compatible.

The `TPersistent` implementation of `Assign` does nothing but calling the `AssignTo` ([446](#)) method of source. This means that if the destination class does not know how to assign the contents of the source class, the source class instance is asked to assign itself to the destination class. This means that it is necessary to implement only one of the two methods so that two classes can be assigned to one another.

Remark In general, a statement of the form

```
Destination:=Source;
```

(where `Destination` and `Source` are classes) does not achieve the same as a statement of the form

```
Destination.Assign(Source);
```

After the former statement, both `Source` and `Destination` will point to the same object. The latter statement will copy the *contents* of the `Source` class to the `Destination` class.

See also: `AssignTo` ([446](#))

4.65.6 TPersistent.FPOAttachObserver

Synopsis: Add an observer to the list of observers.

Declaration: `procedure FPOAttachObserver(AObserver: TObject)`

Visibility: public

Description: `FPOAttachObserver` is part of the implementation of the `IFPObserved` (323) interface in `TPersistent`. It adds a new observer to the list of observers. Calling this multiple times will add the observed object multiple times to the list.

Errors: An `EObserver` exception may be raised if `AObject` does not implement the `IFPObserver` (325) interface.

See also: `IFPObserver` (325), `IFPObserved.FPOAttachObserver` (324), `IFPObserved` (323)

4.65.7 TPersistent.FPODetachObserver

Synopsis: Remove an observer from the list of observers.

Declaration: `procedure FPODetachObserver(AObserver: TObject)`

Visibility: public

Description: `FPODetachObserver` is part of the implementation of the `IFPObserved` (323) interface in `TPersistent`. It removes the first found instance of the observer from the list of observers.

See also: `IFPObserved` (323), `IFPObserved.FPODetachObserver` (324), `IFPObserver` (325)

4.65.8 TPersistent.FPONotifyObservers

Synopsis: Notify observers of changes.

Declaration: `procedure FPONotifyObservers(ASender: TObject;
AOperation: TFPObservedOperation;
Data: Pointer)`

Visibility: public

Description: `FPONotifyObservers` can be called to notify observers of changes in the object. This method simply passes on the parameters that it receives to all attached `IFPObserver` (325) interfaces.

`TPersistent` does not call `FPONotifyObservers`. It is implemented for the benefit of descendant classes.

See also: `IFPObserved` (323), `IFPObserved.FPONotifyObservers` (324), `IFPObserver` (325)

4.65.9 TPersistent.GetNamePath

Synopsis: Returns a string that can be used to identify the class instance.

Declaration: `function GetNamePath : string; Virtual`

Visibility: public

Description: `GetNamePath` returns a string that can be used to identify the class instance. This can be used to display a name for this instance in a Object designer.

`GetNamePath` constructs a name by recursively prepending the `Classname` of the Owner instance to the `Classname` of this instance, separated by a dot.

See also: `TPersistent.GetOwner` (446)

4.66 TProxyStream

4.66.1 Description

TProxyStream is a proxy class for the #rtl.types.IStream (2055) interface. It implements all stream methods by relaying them to the IStream interface.

See also: #rtl.types.IStream (2055), TStreamAdapter (476)

4.66.2 Method overview

Page	Method	Description
450	Check	Check errors.
449	Create	Create a new instance of the TProxyStream class.
449	Read	
449	Seek	
449	Write	

4.66.3 TProxyStream.Create

Synopsis: Create a new instance of the TProxyStream class.

Declaration: `constructor Create(const Stream: IStream)`

Visibility: public

Description: Create initializes a new instance of the TProxyStream class. It saves var stream for use in the other methods.

See also: #rtl.types.IStream (2055)

4.66.4 TProxyStream.Read

Declaration: `function Read(var Buffer; Count: LongInt) : LongInt; Override`

Visibility: public

4.66.5 TProxyStream.Write

Declaration: `function Write(const Buffer; Count: LongInt) : LongInt; Override`

Visibility: public

4.66.6 TProxyStream.Seek

Declaration: `function Seek(const Offset: Int64; Origin: TSeekOrigin) : Int64; Override`

Visibility: public

4.66.7 TProxyStream.Check

Synopsis: Check errors.

Declaration: `procedure Check(err: Integer); Virtual`

Visibility: `public`

Description: `Check` will check the result of the `IStream` interface. This method must be overridden by descendant classes to return interface-specific errors.

See also: `#rtl.types.IStream` (2055)

4.67 TRawByteStringStream

4.67.1 Description

`TRawByteStringStream` is a `TBytesStream` (375) descendent which introduces some auxiliary methods that help with inserting or extracting single-byte strings from the data: `DataStream` (450), `ReadString` (451) and `WriteString` (451).

In difference with `TStringStream` (514), the data has no codepage information associated with it, so the data is always considered to use the default codepage.

See also: `TStringStream` (514), `DataStream` (450), `ReadString` (451), `WriteString` (451)

4.67.2 Method overview

Page	Method	Description
450	<code>Create</code>	Create a new instance of <code>TRawByteStringStream</code> .
450	<code>DataStream</code>	Return the stream contents as a single-byte string.
451	<code>ReadString</code>	Read a string from the stream.
451	<code>WriteString</code>	Write a string to the stream.

4.67.3 TRawByteStringStream.Create

Synopsis: Create a new instance of `TRawByteStringStream`.

Declaration: `constructor Create(const aData: RawByteString); Overload`

Visibility: `public`

Description: `Create` creates a new instance of `TRawByteStringStream` and initializes the `Memory` (400) with the characters in `aData`.

See also: `Memory` (400), `DataStream` (450)

4.67.4 TRawByteStringStream.DataString

Synopsis: Return the stream contents as a single-byte string.

Declaration: `function DataString : RawByteString`

Visibility: `public`

Description: `DataStream` returns the data in `Memory` (400) as a single-byte string without codepage information associated with it. The operation does not change the current position of the stream.

See also: `Memory` (400), `ReadString` (451), `WriteString` (451)

4.67.5 TRawByteStringStream.ReadString

Synopsis: Read a string from the stream.

Declaration: `function ReadString(Count: LongInt) : RawByteString`

Visibility: public

Description: `ReadString` reads at most `Count` bytes from the stream and returns the data as a single-byte string. The operation forwards the current position of the stream with at most `Count` bytes.

See also: `DataString` ([450](#)), `WriteString` ([451](#))

4.67.6 TRawByteStringStream.WriteString

Synopsis: Write a string to the stream.

Declaration: `procedure WriteString(const AString: RawByteString)`

Visibility: public

Description: `WriteString` writes the bytes in `aString` to the stream. The operation forwards the current position of the stream with the length of the stream.

See also: `DataString` ([450](#)), `WriteString` ([451](#))

4.68 TReader

4.68.1 Description

The `TReader` class is a reader class that implements generic component streaming capabilities, independent of the format of the data in the stream. It uses a driver class `TAbstractObjectReader` ([335](#)) to do the actual reading of data. The interface of the `TReader` class should be identical to the interface in Delphi.

Note that the `TReader` design is such that it can read a single component from a stream. It will read all children of this component, but it is not designed to read multiple components in succession from one stream.

It should never be necessary to create an instance of this class directly. Instead, the `TStream.ReadComponent` ([470](#)) call should be used.

See also: `TFile` ([403](#)), `TWriter` ([537](#)), `TAbstractObjectReader` ([335](#))

4.68.2 Method overview

Page	Method	Description
454	BeginReferences	Initializes the component referencing mechanism.
454	CheckValue	Raises an exception if the next value in the stream is not of type Value.
460	CopyValue	Copy a value to a writer.
453	Create	Creates a new reader class.
454	DefineBinaryProperty	Reads a user-defined binary property from the stream.
454	DefineProperty	Reads a user-defined property from the stream.
453	Destroy	Destroys a reader class.
454	EndOfList	Returns true if the stream contains an end-of-list marker.
455	EndReferences	Finalizes the component referencing mechanism.
455	FixupReferences	Tries to resolve all unresolved component references.
453	FlushBuffer	Flush the buffer.
455	NextValue	Returns the type of the next value.
455	Read	Read raw data from stream.
455	ReadBoolean	Reads a boolean from the stream.
456	ReadChar	Reads a character from the stream.
456	ReadCollection	Reads a collection from the stream.
456	ReadComponent	Starts reading a component from the stream.
457	ReadComponents	Starts reading child components from the stream.
457	ReadCurrency	Read a currency value from the stream.
457	ReadDate	Reads a date from the stream.
457	ReadDouble	
457	ReadFloat	Reads a float from the stream.
458	ReadIdent	Reads an identifier from the stream.
458	ReadInt64	Reads a 64-bit integer from the stream.
458	ReadInteger	Reads an integer from the stream.
458	ReadListBegin	Checks for the beginning of a list.
459	ReadListEnd	Checks for the end of a list.
459	ReadRootComponent	Starts reading a root component.
458	ReadSet	Read a set value from the stream.
459	ReadSignature	Read stream signature from the stream.
457	ReadSingle	Reads a single-type real from the stream.
459	ReadString	Reads a string from the stream.
456	ReadUnicodeChar	Read Unicode character.
460	ReadUnicodeString	Read a UnicodeString value from the stream.
460	ReadValue	Reads the next value type from the stream.
459	ReadVariant	Read a variant from the stream.
456	ReadWideChar	Read widechar from the stream.
460	ReadWideString	Read a WideString value from the stream.
455	SkipValue	

4.68.3 Property overview

Page	Properties	Access	Description
460	Driver	r	The driver in use for streaming the data.
462	OnAncestorNotFound	rw	Handler called when the ancestor component cannot be found.
463	OnCreateComponent	rw	Handler called when a component needs to be created.
461	OnError	rw	Handler called when an error occurs.
463	OnFindComponentClass	rw	Handler called when a component class reference needs to be found.
462	OnFindMethod	rw	Handler to find or change a method address.
461	OnPropertyNotFound	rw	Handler for treating missing properties.
463	OnReadStringProperty	rw	Handler for translating strings when read from the stream.
462	OnReferenceName	rw	Handler called when another component is referenced.
462	OnSetMethodProperty	rw	Handler for setting method properties.
462	OnSetName	rw	Handler called when setting a component name.
461	Owner	rw	Owner of the component being read.
461	Parent	rw	Parent of the component being read.

4.68.4 TReader.Create

Synopsis: Creates a new reader class.

Declaration: `constructor Create(Stream: TStream; BufSize: Integer)`

Visibility: `public`

Description: Creates a new reader class.

4.68.5 TReader.Destroy

Synopsis: Destroys a reader class.

Declaration: `destructor Destroy; Override`

Visibility: `public`

Description: Destroys a reader class.

4.68.6 TReader.FlushBuffer

Synopsis: Flush the buffer.

Declaration: `procedure FlushBuffer; Override`

Visibility: `public`

Description: `FlushBuffer` flushes the buffer. It is provided for Delphi compatibility, and is not used in FPC.

See also: `TFile.FlushBuffer` ([404](#)), `TAbstractObjectReader.FlushBuffer` ([336](#))

4.68.7 TReader.BeginReferences

Synopsis: Initializes the component referencing mechanism.

Declaration: `procedure BeginReferences`

Visibility: `public`

Description: When streaming components, the streaming mechanism keeps a list of existing components that can be referenced to. This method initializes up that system.

4.68.8 TReader.CheckValue

Synopsis: Raises an exception if the next value in the stream is not of type `Value`.

Declaration: `procedure CheckValue(Value: TValueType)`

Visibility: `public`

Description: Raises an exception if the next value in the stream is not of type `Value`.

4.68.9 TReader.DefineProperty

Synopsis: Reads a user-defined property from the stream.

Declaration: `procedure DefineProperty(const Name: string; AReadData: TReaderProc;
WriteData: TWriterProc; HasData: Boolean)
; Override`

Visibility: `public`

Description: Reads a user-defined property from the stream.

4.68.10 TReader.DefineBinaryProperty

Synopsis: Reads a user-defined binary property from the stream.

Declaration: `procedure DefineBinaryProperty(const Name: string;
AReadData: TStreamProc;
WriteData: TStreamProc; HasData: Boolean)
; Override`

Visibility: `public`

Description: Reads a user-defined binary property from the stream.

4.68.11 TReader.EndOfList

Synopsis: Returns true if the stream contains an end-of-list marker.

Declaration: `function EndOfList : Boolean`

Visibility: `public`

Description: Returns true if the stream contains an end-of-list marker.

4.68.12 TReader.EndReferences

Synopsis: Finalizes the component referencing mechanism.

Declaration: `procedure EndReferences`

Visibility: `public`

Description: When streaming components, the streaming mechanism keeps a list of existing components that can be referenced to. This method cleans up that system.

4.68.13 TReader.FixupReferences

Synopsis: Tries to resolve all unresolved component references.

Declaration: `procedure FixupReferences`

Visibility: `public`

Description: Tries to resolve all unresolved component references.

4.68.14 TReader.SkipValue

Declaration: `procedure SkipValue`

Visibility: `public`

4.68.15 TReader.NextValue

Synopsis: Returns the type of the next value.

Declaration: `function NextValue : TValueType`

Visibility: `public`

Description: Returns the type of the next value.

4.68.16 TReader.Read

Synopsis: Read raw data from stream.

Declaration: `procedure Read(var Buf; Count: LongInt); Virtual`

Visibility: `public`

Description: `Read` is introduced for Delphi compatibility to read raw data from the component stream. This should not be used in new production code as it will totally mess up the streaming.

See also: `TAbstractObjectReader.Read` ([338](#)), `TBinaryObjectReader.Read` ([358](#))

4.68.17 TReader.ReadBoolean

Synopsis: Reads a boolean from the stream.

Declaration: `function ReadBoolean : Boolean`

Visibility: `public`

Description: Reads a boolean from the stream.

4.68.18 TReader.ReadChar

Synopsis: Reads a character from the stream.

Declaration: `function ReadChar : Char`

Visibility: `public`

Description: Reads a character from the stream.

4.68.19 TReader.ReadWideChar

Synopsis: Read widechar from the stream.

Declaration: `function ReadWideChar : WideChar`

Visibility: `public`

Description: `TReader.ReadWideChar` reads a widechar from the stream. This actually reads a widestring and returns the first character.

See also: `TReader.ReadWideString` ([460](#)), `TWriter.WriteWideChar` ([540](#))

4.68.20 TReader.ReadUnicodeChar

Synopsis: Read Unicode character.

Declaration: `function ReadUnicodeChar : UnicodeChar`

Visibility: `public`

Description: `ReadUnicodeChar` reads a single Unicode character from the stream. It does this by reading a `UnicodeString` string from the stream and returning the first character.

Errors: If the string has a length different from 1, an `EReadError` exception will occur.

See also: `TReader.ReadUnicodeString` ([460](#))

4.68.21 TReader.ReadCollection

Synopsis: Reads a collection from the stream.

Declaration: `procedure ReadCollection(Collection: TCollection)`

Visibility: `public`

Description: Reads a collection from the stream.

4.68.22 TReader.ReadComponent

Synopsis: Starts reading a component from the stream.

Declaration: `function ReadComponent(Component: TComponent) : TComponent`

Visibility: `public`

Description: Starts reading a component from the stream.

4.68.23 TReader.ReadComponents

Synopsis: Starts reading child components from the stream.

Declaration: `procedure ReadComponents(AOwner: TComponent; AParent: TComponent;
Proc: TReadComponentsProc)`

Visibility: public

Description: Starts reading child components from the stream.

4.68.24 TReader.ReadFloat

Synopsis: Reads a float from the stream.

Declaration: `function ReadFloat : Extended`

Visibility: public

Description: Reads a float from the stream.

4.68.25 TReader.ReadSingle

Synopsis: Reads a single-type real from the stream.

Declaration: `function ReadSingle : Single`

Visibility: public

Description: Reads a single-type real from the stream.

4.68.26 TReader.ReadDouble

Declaration: `function ReadDouble : Double`

Visibility: public

4.68.27 TReader.ReadDate

Synopsis: Reads a date from the stream.

Declaration: `function ReadDate : TDateTime`

Visibility: public

Description: Reads a date from the stream.

4.68.28 TReader.ReadCurrency

Synopsis: Read a currency value from the stream.

Declaration: `function ReadCurrency : Currency`

Visibility: public

Description: `ReadCurrency` reads a currency typed value from the stream and returns the result. This method does nothing except call the driver method of the driver being used.

See also: `TWriter.WriteCurrency` ([541](#))

4.68.29 TReader.ReadIdent

Synopsis: Reads an identifier from the stream.

Declaration: `function ReadIdent : string`

Visibility: `public`

Description: Reads an identifier from the stream.

4.68.30 TReader.ReadInteger

Synopsis: Reads an integer from the stream.

Declaration: `function ReadInteger : LongInt`

Visibility: `public`

Description: Reads an integer from the stream.

4.68.31 TReader.ReadInt64

Synopsis: Reads a 64-bit integer from the stream.

Declaration: `function ReadInt64 : Int64`

Visibility: `public`

Description: Reads a 64-bit integer from the stream.

4.68.32 TReader.ReadSet

Synopsis: Read a set value from the stream.

Declaration: `function ReadSet (EnumType: Pointer) : Integer`

Visibility: `public`

Description: `ReadSet` reads a set of elements with type `EnumType` and returns them as an integer where each element is encoded in a bit of the integer. Thus, at most an enumerated type with 32 elements can be read with this function.

Errors: No checking is performed on the validity of `EnumType`. It is assumed to be a valid `PTypeInfo` pointer.

See also: `TWriter.WriteSet` ([541](#))

4.68.33 TReader.ReadListBegin

Synopsis: Checks for the beginning of a list.

Declaration: `procedure ReadListBegin`

Visibility: `public`

Description: Checks for the beginning of a list.

4.68.34 TReader.ReadListEnd

Synopsis: Checks for the end of a list.

Declaration: `procedure ReadListEnd`

Visibility: `public`

Description: Checks for the end of a list.

4.68.35 TReader.ReadRootComponent

Synopsis: Starts reading a root component.

Declaration: `function ReadRootComponent (ARoot: TComponent) : TComponent`

Visibility: `public`

Description: Starts reading a root component.

4.68.36 TReader.ReadVariant

Synopsis: Read a variant from the stream.

Declaration: `function ReadVariant : Variant`

Visibility: `public`

Description: `ReadVariant` reads the next value from the stream and returns it as a variant. No variant array can be read from the stream, only single values.

Errors: If no variant manager is installed, the function will raise an `EReadError` exception. If the next value is not a simple value, again an `EReadError` exception is raised. exception is

See also: `TBinaryObjectWriter.WriteVariant` ([368](#))

4.68.37 TReader.ReadSignature

Synopsis: Read stream signature from the stream.

Declaration: `procedure ReadSignature`

Visibility: `public`

Description: `ReadSignature` is called when starting to read a root component from a stream. Some streams contain a signature (header) to detect whether the stream contains correct data.

Errors: If the stream does not start with the correct signature, an `EReadError` ([321](#)) exception will be raised.

See also: `TAbstractObjectReader.ReadSignature` ([342](#)), `TAbstractObjectWriter.WriteSignature` ([345](#))

4.68.38 TReader.ReadString

Synopsis: Reads a string from the stream.

Declaration: `function ReadString : string`

Visibility: `public`

Description: Reads a string from the stream.

4.68.39 TReader.ReadWideString

Synopsis: Read a WideString value from the stream.

Declaration: `function ReadWideString : WideString`

Visibility: public

Description: `ReadWideString` reads a `WideString` typed value from the stream and returns the result. This method does nothing except call the driver method of the driver being used.

See also: `TWriter.WriteString` ([543](#))

4.68.40 TReader.ReadUnicodeString

Synopsis: Read a UnicodeString value from the stream.

Declaration: `function ReadUnicodeString : UnicodeString`

Visibility: public

Description: `ReadUnicodeString` reads a `UnicodeString` string from the stream. The stream can contain a string from any type, it will be converted to `UnicodeString`.

See also: `TAbstractObjectReader.ReadUnicodeString` ([343](#)), `TWriter.WriteUnicodeString` ([543](#))

4.68.41 TReader.ReadValue

Synopsis: Reads the next value type from the stream.

Declaration: `function ReadValue : TValueType`

Visibility: public

Description: Reads the next value type from the stream.

4.68.42 TReader.CopyValue

Synopsis: Copy a value to a writer.

Declaration: `procedure CopyValue (Writer: TWriter)`

Visibility: public

Description: `CopyValue` reads the next value from the reader stream, and writes it to the passed `Writer`.

4.68.43 TReader.Driver

Synopsis: The driver in use for streaming the data.

Declaration: `Property Driver : TAbstractObjectReader`

Visibility: public

Access: Read

Description: The driver in use for streaming the data.

4.68.44 TReader.Owner

Synopsis: Owner of the component being read.

Declaration: `Property Owner : TComponent`

Visibility: `public`

Access: `Read,Write`

Description: Owner of the component being read.

4.68.45 TReader.Parent

Synopsis: Parent of the component being read.

Declaration: `Property Parent : TComponent`

Visibility: `public`

Access: `Read,Write`

Description: Parent of the component being read.

4.68.46 TReader.OnError

Synopsis: Handler called when an error occurs.

Declaration: `Property OnError : TReaderError`

Visibility: `public`

Access: `Read,Write`

Description: Handler called when an error occurs.

4.68.47 TReader.OnPropertyNotFound

Synopsis: Handler for treating missing properties.

Declaration: `Property OnPropertyNotFound : TPropertyNotFoundEvent`

Visibility: `public`

Access: `Read,Write`

Description: `OnPropertyNotFound` can be used to take appropriate action when a property is read from a stream and no such property is found in the RTTI information of the Instance that is being read from the stream. It can be set at runtime, or at design time by an IDE.

For more information about the meaning of the various arguments to the event handler, see `TPropertyNotFoundEvent` ([294](#)).

See also: `TPropertyNotFoundEvent` ([294](#)), `TReader.OnSetMethodProperty` ([462](#)), `TReader.OnReadStringProperty` ([463](#))

4.68.48 TReader.OnFindMethod

Synopsis: Handler to find or change a method address.

Declaration: `Property OnFindMethod : TFindMethodEvent`

Visibility: `public`

Access: `Read,Write`

Description: Handler to find or change a method address.

4.68.49 TReader.OnSetMethodProperty

Synopsis: Handler for setting method properties.

Declaration: `Property OnSetMethodProperty : TSetMethodPropertyEvent`

Visibility: `public`

Access: `Read,Write`

Description: `OnSetMethodProperty` can be set to handle the setting of method properties. This handler can be used by an IDE to prevent methods from actually being assigned when an object is being streamed in the designer.

See also: `TReader.OnReadStringProperty` ([463](#)), `TReader.OnPropertyNotFound` ([461](#))

4.68.50 TReader.OnSetName

Synopsis: Handler called when setting a component name.

Declaration: `Property OnSetName : TSetNameEvent`

Visibility: `public`

Access: `Read,Write`

Description: Handler called when setting a component name.

4.68.51 TReader.OnReferenceName

Synopsis: Handler called when another component is referenced.

Declaration: `Property OnReferenceName : TReferenceNameEvent`

Visibility: `public`

Access: `Read,Write`

Description: Handler called when another component is referenced.

4.68.52 TReader.OnAncestorNotFound

Synopsis: Handler called when the ancestor component cannot be found.

Declaration: `Property OnAncestorNotFound : TAncestorNotFoundEvent`

Visibility: `public`

Access: `Read,Write`

Description: Handler called when the ancestor component cannot be found.

4.68.53 TReader.OnCreateComponent

Synopsis: Handler called when a component needs to be created.

Declaration: Property OnCreateComponent : TCreateComponentEvent

Visibility: public

Access: Read,Write

Description: Handler called when a component needs to be created.

4.68.54 TReader.OnFindComponentClass

Synopsis: Handler called when a component class reference needs to be found.

Declaration: Property OnFindComponentClass : TFindComponentClassEvent

Visibility: public

Access: Read,Write

Description: Handler called when a component class reference needs to be found.

4.68.55 TReader.OnReadStringProperty

Synopsis: Handler for translating strings when read from the stream.

Declaration: Property OnReadStringProperty : TReadWriteStringPropertyEvent

Visibility: public

Access: Read,Write

Description: OnReadStringProperty is called whenever a string property is read from the stream. It can be used e.g. by a translation mechanism to translate the strings on the fly, when a form is loaded. See TReadWriteStringPropertyEvent (294) for a description of the various parameters.

See also: TReader.OnPropertyNotFound (461), TReader.OnSetMethodProperty (462), TReadWriteStringPropertyEvent (294)

4.69 TRecall

4.69.1 Description

TRecall is a helper class used to copy published properties of a class (the reference object) in another class (the storage object). The reference object and storage object must be assignable to each other.

The TRecall can be used to store the state of a persistent class, and restore it at a later time.

When a TRecall object is created, it gets passed a reference instance and a storage instance. It immediately stores the properties of the reference object in the storage object.

The Store (464) method can be called throughout the lifetime of the reference object to update the stored properties.

When the TRecall instance is destroyed then the properties are copied from the storage object to the reference object. The storage object is freed automatically.

If the properties should not be copied back from the storage to the reference object, the Forget (465) can be called.

See also: `TRecall.Create` (464), `TRecall.Destroy` (464), `TRecall.Forget` (465), `TRecall.Store` (464), `TPersistent.Assign` (447)

4.69.2 Method overview

Page	Method	Description
464	Create	Creates a new instance of <code>TRecall</code> .
464	Destroy	Copies the stored properties to the reference object and destroys the <code>TRecall</code> instance.
465	Forget	Clear the reference property.
464	Store	Assigns the reference instance to the storage instance.

4.69.3 Property overview

Page	Properties	Access	Description
465	Reference	r	The reference object.

4.69.4 TRecall.Create

Synopsis: Creates a new instance of `TRecall`.

Declaration: `constructor Create(AStorage: TPersistent; AReference: TPersistent)`

Visibility: `public`

Description: `Create` creates a new instance of `TRecall` and initializes the Reference and Storage instances. It calls `Store` (464) to assign the reference object properties to the storage instance.

See also: `TRecall.Store` (464), `TRecall.Destroy` (464)

4.69.5 TRecall.Destroy

Synopsis: Copies the stored properties to the reference object and destroys the `TRecall` instance.

Declaration: `destructor Destroy; Override`

Visibility: `public`

Description: `Destroy` assigns the storage instance to the reference instance, if the latter is still valid. After this, it frees the storage and calls the inherited `destroy`.

Errors: `Destroy` does not check whether the reference (465) instance is still valid. If the reference pointer was invalidated, call `TRecall.Forget` (465) to clear the reference instance.

See also: `TRecall.Store` (464), `TRecall.Forget` (465)

4.69.6 TRecall.Store

Synopsis: Assigns the reference instance to the storage instance.

Declaration: `procedure Store`

Visibility: `public`

Description: `Store` assigns the reference instance to the storage instance. This will only work if the two classes can be assigned to each other.

This method can be used to refresh the storage.

Errors: `Store` does not check whether the reference (465) instance is still valid. If the reference pointer was invalidated, call `TRecall.Forget` (465) to clear the reference instance.

4.69.7 TRecall.Forget

Synopsis: Clear the reference property.

Declaration: `procedure Forget`

Visibility: `public`

Description: `Forget` sets the `Reference` (465) property to `Nil`. When the `TRecall` instance is destroyed, the reference instance will not be restored.

Note that after a call to `Forget`, a call to `Store` (464) has no effect.

Errors: None.

See also: `TRecall.Reference` (465), `TRecall.Store` (464), `TRecall.Destroy` (464)

4.69.8 TRecall.Reference

Synopsis: The reference object.

Declaration: `Property Reference : TPersistent`

Visibility: `public`

Access: `Read`

Description: `Reference` is the instance of the reference object. Do not free the reference directly. Call `Forget` (465) to clear the reference and then free the reference object.

See also: `TRecall.Forget` (465)

4.70 TResourceStream

4.70.1 Description

Stream that reads its data from a resource object.

4.70.2 Method overview

Page	Method	Description
466	<code>Create</code>	Creates a new instance of a resource stream.
466	<code>CreateFromID</code>	Creates a new instance of a resource stream with a resource.
466	<code>Destroy</code>	Destroys the instance of the resource stream.

4.70.3 TResourceStream.Create

Synopsis: Creates a new instance of a resource stream.

Declaration: constructor `Create`(Instance: TFPResourceHMODULE; const ResName: string;
ResType: PChar)

Visibility: public

Description: Creates a new instance of a resource stream.

4.70.4 TResourceStream.CreateFromID

Synopsis: Creates a new instance of a resource stream with a resource.

Declaration: constructor `CreateFromID`(Instance: TFPResourceHMODULE; ResID: Integer;
ResType: PChar)

Visibility: public

Description: The resource is loaded from the loaded module (identified by the handle `Instance`), identifier `ResID` and type `ResType`.

4.70.5 TResourceStream.Destroy

Synopsis: Destroys the instance of the resource stream.

Declaration: destructor `Destroy`; Override

Visibility: public

Description: Destroys the instance of the resource stream.

4.71 TStream

4.71.1 Description

`TStream` is the base class for all streaming classes. It defines methods for reading (467), writing (468) from and to streams, as well as functions to determine the size of the stream as well as the current position of the stream.

Descendant classes such as `TMemoryStream` (436) or `TFileStream` (406) then override these methods to write streams to memory or file.

See also: `TMemoryStream` (436), `TFileStream` (406), `TStringStream` (514)

4.71.2 Method overview

Page	Method	Description
470	<code>CopyFrom</code>	Copy data from one stream to another.
472	<code>FixupResourceHeader</code>	Not implemented in FPC.
467	<code>Read</code>	Reads data from the stream to a buffer and returns the number of bytes read.
474	<code>ReadAnsiString</code>	Read an ansistring from the stream and return its value.
469	<code>ReadBuffer</code>	Reads data from the stream to a buffer.
472	<code>ReadByte</code>	Read a byte from the stream and return its value.
470	<code>ReadComponent</code>	Reads component data from a stream.
470	<code>ReadComponentRes</code>	Reads component data and resource header from a stream.
473	<code>ReadDWord</code>	Read a DWord from the stream and return its value.
473	<code>ReadQWord</code>	Read a QWord value from the stream and return its value.
472	<code>ReadResHeader</code>	Read a resource header from the stream.
473	<code>ReadWord</code>	Read a word from the stream and return its value.
468	<code>Seek</code>	Sets the current position in the stream.
468	<code>Write</code>	Writes data from a buffer to the stream and returns the number of bytes written.
475	<code>WriteAnsiString</code>	Write an ansistring to the stream.
469	<code>WriteBuffer</code>	Writes data from a buffer to the stream.
474	<code>WriteByte</code>	Write a byte to the stream.
471	<code>WriteComponent</code>	Write component data to the stream.
471	<code>WriteComponentRes</code>	Write resource header and component data to a stream.
471	<code>WriteDescendent</code>	Write descendent of a component.
471	<code>WriteDescendentRes</code>	Write descendent of a component as resource.
474	<code>WriteDWord</code>	Write a DWord to the stream.
475	<code>WriteQWord</code>	Write a QWord value to the stream.
472	<code>WriteResourceHeader</code>	Write resource header to the stream.
474	<code>WriteWord</code>	Write a word to the stream.

4.71.3 Property overview

Page	Properties	Access	Description
475	<code>Position</code>	rw	The current position in the stream.
476	<code>Size</code>	rw	The current size of the stream.

4.71.4 TStream.Read

Synopsis: Reads data from the stream to a buffer and returns the number of bytes read.

Declaration: `function Read(var Buffer; Count: LongInt) : LongInt; Virtual; Overload`

Visibility: `public`

Description: `Read` attempts to read `Count` from the stream to `Buffer` and returns the number of bytes actually read.

This method should be used when the number of bytes is not determined. If a specific number of bytes is expected, use `TStream.ReadBuffer` ([469](#)) instead.

As implemented in `TStream`, `Read` does nothing but raises an `EStreamError` ([321](#)) exception to indicate that reading is not supported. Descendant classes that allow reading must override this method to do the actual reading.

Descendant classes should (if they don't explicitly raise an exception) return a positive value (≥ 0), where zero indicates an error.

Errors: In case a descendant class does not allow reading from the stream, an exception is raised.

See also: `TStream.Write` (468), `TStream.ReadBuffer` (469)

4.71.5 TStream.Write

Synopsis: Writes data from a buffer to the stream and returns the number of bytes written.

Declaration: `function Write(const Buffer; Count: LongInt) : LongInt; Virtual
; Overload`

Visibility: public

Description: `Write` attempts to write `Count` bytes from `Buffer` to the stream. It returns the actual number of bytes written to the stream.

This method should be used when the number of bytes that should be written is not determined. If a specific number of bytes should be written, use `TStream.WriteBuffer` (469) instead.

As implemented in `TStream`, `Write` does nothing but raises `EStreamError` (321) exception to indicate that writing is not supported. Descendant classes that allow writing must override this method to do the actual writing.

Descendant classes should (if they don't explicitly raise an exception) return a positive value (≥ 0), where zero indicates an error.

Errors: In case a descendant class does not allow writing to the stream, an exception is raised.

See also: `TStream.Read` (467), `TStream.WriteBuffer` (469)

4.71.6 TStream.Seek

Synopsis: Sets the current position in the stream.

Declaration: `function Seek(Offset: LongInt; Origin: Word) : LongInt; Virtual
; Overload
function Seek(const Offset: Int64; Origin: TSeekOrigin) : Int64
; Virtual; Overload`

Visibility: public

Description: `Seek` sets the position of the stream to `Offset` bytes from `Origin`. There is a 32-bit variant of this function and a 64-bit variant. The difference can be made by choosing the correct `Offset` parameter: the integer-typed parameter selects the 32-bit variant, the parameter of type `TSeekOrigin` (294) selects the 64-bit variant of the function.

Both variants of this function return the new (0-based) position in the stream.

The `Origin` parameter for the 32-bit version can have one of the following values:

Table 4.30:

Constant	Meaning
<code>soFromBeginning</code>	Set the position relative to the start of the stream.
<code>soFromCurrent</code>	Set the position relative to the current position in the stream.
<code>soFromEnd</code>	Set the position relative to the end of the stream.

These values are defined in the `SysUtils` (281) unit.

The `Origin` parameter for the 64-bit version has one of the following values:

Table 4.31:

Value	Meaning
<code>soBeginning</code>	Offset is interpreted relative to the start of the stream.
<code>soCurrent</code>	Offset is interpreted relative to the current position in the stream.
<code>soEnd</code>	Offset is interpreted relative to the end of the stream.

Offset should be negative when the origin is `SoFromEnd` (`soEnd`). It should be positive for `soFromBeginning` and can have both signs for `soFromCurrent`.

This is an abstract method, which must be overridden by descendant classes. They may choose not to implement this method for all values of `Origin` and `Offset`.

Remark Internally, all calls are re-routed to the 64-bit version of the call. When creating a descendant of `TStream`, the 64-bit version of the call should be overridden.

Errors: An exception may be raised if this method is called with an invalid pair of `Offset, Origin` values. e.g. a negative offset for `soFromBeginning` (or `soBeginning`).

See also: `TStream.Position` (475)

4.71.7 TStream.ReadBuffer

Synopsis: Reads data from the stream to a buffer.

Declaration: `procedure ReadBuffer(var Buffer; Count: LongInt)`

Visibility: public

Description: `ReadBuffer` reads `Count` bytes of the stream into `Buffer`. If the stream does not contain `Count` bytes, then an exception is raised.

`ReadBuffer` should be used to read in a fixed number of bytes, such as when reading structures or the content of variables. If the number of bytes is not determined, use `TStream.Read` (467) instead. `ReadBuffer` uses `Read` internally to do the actual reading.

Errors: If the stream does not allow to read `Count` bytes, then an exception is raised.

See also: `TStream.Read` (467), `TStream.WriteBuffer` (469)

4.71.8 TStream.WriteBuffer

Synopsis: Writes data from a buffer to the stream.

Declaration: `procedure WriteBuffer(const Buffer; Count: LongInt)`

Visibility: public

Description: `WriteBuffer` writes `Count` bytes to the stream from `Buffer`. If the stream does not allow `Count` bytes to be written, then an exception is raised.

`WriteBuffer` should be used to write a fixed number of bytes, such as when writing structures or the content of variables. If the number of bytes is not determined, use `TStream.Write` (468) instead. `WriteBuffer` uses `Write` internally to do the actual writing.

Errors: If the stream does not allow to write `Count` bytes, then an exception is raised.

See also: `TStream.Write` (468), `TStream.ReadBuffer` (469)

4.71.9 TStream.CopyFrom

Synopsis: Copy data from one stream to another.

Declaration: `function CopyFrom(Source: TStream; Count: Int64) : Int64`

Visibility: public

Description: `CopyFrom` reads `Count` bytes from `Source` and writes them to the current stream. This updates the current position in the stream. After the action is completed, the number of bytes copied is returned. If `Count` is zero, then the whole contents of the `Source` stream is copied. It is positioned on the first byte of data, and `Size` bytes are copied. Note that this cannot be used with streams that do not allow seeking or do not allow determining the size of the stream.

This can be used to quickly copy data from one stream to another or to copy the whole contents of the stream.

See also: `TStream.Read` (467), `TStream.Write` (468)

4.71.10 TStream.ReadComponent

Synopsis: Reads component data from a stream.

Declaration: `function ReadComponent(Instance: TComponent) : TComponent`

Visibility: public

Description: `ReadComponent` reads a component state from the stream and transfers this state to `Instance`. If `Instance` is `nil`, then it is created first based on the type stored in the stream. `ReadComponent` returns the component as it is read from the stream.

`ReadComponent` simply creates a `TReader` (451) object and calls its `ReadRootComponent` (459) method.

Errors: If an error occurs during the reading of the component, an `EFileError` (319) exception is raised.

See also: `TStream.WriteComponent` (471), `TStream.ReadComponentRes` (470), `TReader.ReadRootComponent` (459)

4.71.11 TStream.ReadComponentRes

Synopsis: Reads component data and resource header from a stream.

Declaration: `function ReadComponentRes(Instance: TComponent) : TComponent`

Visibility: public

Description: `ReadComponentRes` reads a resource header from the stream, and then calls `ReadComponent` (470) to read the component state from the stream into `Instance`.

This method is usually called by the global streaming method when instantiating forms and datamodules as created by an IDE. It should be used mainly on Windows, to store components in Windows resources.

Errors: If an error occurs during the reading of the component, an `EFileError` (319) exception is raised.

See also: `TStream.ReadComponent` (470), `TStream.WriteComponentRes` (471)

4.71.12 TStream.WriteComponent

Synopsis: Write component data to the stream.

Declaration: `procedure WriteComponent (Instance: TComponent)`

Visibility: public

Description: `WriteComponent` writes the published properties of `Instance` to the stream, so they can later be read with `TStream.ReadComponent` (470). This method is intended to be used by an IDE, to preserve the state of a form or datamodule as designed in the IDE.

`WriteComponent` simply calls `WriteDescendant` (466) with `Nil` ancestor.

See also: `TStream.ReadComponent` (470), `TStream.WriteComponentRes` (471)

4.71.13 TStream.WriteComponentRes

Synopsis: Write resource header and component data to a stream.

Declaration: `procedure WriteComponentRes (const ResName: string; Instance: TComponent)`

Visibility: public

Description: `WriteComponentRes` writes a `ResName` resource header to the stream and then calls `WriteComponent` (471) to write the published properties of `Instance` to the stream.

This method is intended for use by an IDE that can use it to store forms or datamodules as designed in a Windows resource stream.

See also: `TStream.WriteComponent` (471), `TStream.ReadComponentRes` (470)

4.71.14 TStream.WriteDescendent

Synopsis: Write descendent of a component.

Declaration: `procedure WriteDescendent (Instance: TComponent; Ancestor: TComponent)`

Visibility: public

Description: `WriteDescendent` will create a `TWriter` (537) writer class and write `Instance` as a descendent of `Ancestor` using the writer. This is used to create diff streams: only the properties where `Instance` differs from `Ancestor` are written to the stream.

See also: `TWriter.WriteDescendent` (540)

4.71.15 TStream.WriteDescendentRes

Synopsis: Write descendent of a component as resource.

Declaration: `procedure WriteDescendentRes (const ResName: string;
Instance: TComponent; Ancestor: TComponent)`

Visibility: public

Description: `WriteDescendentRes` calls `WriteDescendent` as a resource stream.

See also: `TWriter.WriteDescendent` (540)

4.71.16 TStream.WriteResourceHeader

Synopsis: Write resource header to the stream.

Declaration: `procedure WriteResourceHeader(const ResName: string;
var FixupInfo: LongInt)`

Visibility: public

Description: `WriteResourceHeader` writes a resource-file header for a resource called `ResName`. It returns in `FixupInfo` the argument that should be passed on to `TStream.FixupResourceHeader` (472).

`WriteResourceHeader` should not be used directly. It is called by the `TStream.WriteComponentRes` (471) and `TStream.WriteDescendantRes` (466) methods.

See also: `TStream.FixupResourceHeader` (472), `TStream.WriteComponentRes` (471), `TStream.WriteDescendantRes` (466)

4.71.17 TStream.FixupResourceHeader

Synopsis: Not implemented in FPC.

Declaration: `procedure FixupResourceHeader(FixupInfo: LongInt)`

Visibility: public

Description: `FixupResourceHeader` is used to write the size of the resource after a component was written to stream. The size is determined from the current position, and it is written at position `FixupInfo`. After that the current position is restored.

`FixupResourceHeader` should never be called directly; it is handled by the streaming system.

See also: `TStream.WriteResourceHeader` (472), `TStream.WriteComponentRes` (471), `TStream.WriteDescendantRes` (466)

4.71.18 TStream.ReadResHeader

Synopsis: Read a resource header from the stream.

Declaration: `procedure ReadResHeader`

Visibility: public

Description: `ReadResourceHeader` reads a resource file header from the stream. It positions the stream just beyond the header.

`ReadResourceHeader` should not be called directly, it is called by the streaming system when needed.

Errors: If the resource header is invalid an `EInvalidImage` (320) exception is raised.

See also: `TStream.ReadComponentRes` (470), `EInvalidImage` (320)

4.71.19 TStream.ReadByte

Synopsis: Read a byte from the stream and return its value.

Declaration: `function ReadByte : Byte`

Visibility: public

Description: `ReadByte` reads one byte from the stream and returns its value.

Errors: If the byte cannot be read, a `EStreamError` (321) exception will be raised. This is a utility function which simply calls the `Read` (467) function.

See also: `TStream.Read` (467), `TStream.WriteByte` (474), `TStream.ReadWord` (473), `TStream.ReadDWord` (473), `TStream.ReadAnsiString` (474)

4.71.20 `TStream.ReadWord`

Synopsis: Read a word from the stream and return its value.

Declaration: `function ReadWord : Word`

Visibility: `public`

Description: `ReadWord` reads one `Word` (i.e. 2 bytes) from the stream and returns its value. This is a utility function which simply calls the `Read` (467) function.

Errors: If the word cannot be read, a `EStreamError` (321) exception will be raised.

See also: `TStream.Read` (467), `TStream.WriteWord` (474), `TStream.ReadByte` (472), `TStream.ReadDWord` (473), `TStream.ReadAnsiString` (474)

4.71.21 `TStream.ReadDWord`

Synopsis: Read a `DWord` from the stream and return its value.

Declaration: `function ReadDWord : Cardinal`

Visibility: `public`

Description: `ReadDWord` reads one `DWord` (i.e. 4 bytes) from the stream and returns its value. This is a utility function which simply calls the `Read` (467) function.

Errors: If the `DWord` cannot be read, a `EStreamError` (321) exception will be raised.

See also: `TStream.Read` (467), `TStream.WriteDWord` (474), `TStream.ReadByte` (472), `TStream.ReadWord` (473), `TStream.ReadAnsiString` (474)

4.71.22 `TStream.ReadQWord`

Synopsis: Read a `QWord` value from the stream and return its value.

Declaration: `function ReadQWord : QWord`

Visibility: `public`

Description: `ReadQWord` reads a `QWord` value (8 bytes) from the stream and returns its value.

Errors: If not enough bytes are available on the stream, an `EStreamError` (321) exception will be raised.

See also: `TStream.Read` (467), `TStream.WriteByte` (474), `TStream.ReadWord` (473), `TStream.ReadDWord` (473), `TStream.ReadAnsiString` (474)

4.71.23 TStream.ReadAnsiString

Synopsis: Read an ansistring from the stream and return its value.

Declaration: `function ReadAnsiString : string`

Visibility: `public`

Description: `ReadAnsiString` reads an ansistring from the stream and returns its value. This is a utility function which simply calls the read function several times. The Ansistring should be stored as 4 bytes (a DWord) representing the length of the string, and then the string value itself. The `WriteAnsiString` (475) function writes an ansistring in such a format.

Errors: If the AnsiString cannot be read, a `EStreamError` (321) exception will be raised.

See also: `TStream.Read` (467), `TStream.WriteAnsiString` (475), `TStream.ReadByte` (472), `TStream.ReadWord` (473), `TStream.ReadDWord` (473)

4.71.24 TStream.WriteByte

Synopsis: Write a byte to the stream.

Declaration: `procedure WriteByte(b: Byte)`

Visibility: `public`

Description: `WriteByte` writes the byte B to the stream. This is a utility function which simply calls the `Write` (468) function. The byte can be read from the stream using the `ReadByte` (472) function.

Errors: If an error occurs when attempting to write, an `EStreamError` (321) exception will be raised.

See also: `TStream.Write` (468), `TStream.ReadByte` (472), `TStream.WriteWord` (474), `TStream.WriteDWord` (474), `TStream.WriteAnsiString` (475)

4.71.25 TStream.WriteWord

Synopsis: Write a word to the stream.

Declaration: `procedure WriteWord(w: Word)`

Visibility: `public`

Description: `WriteWord` writes the word W (i.e. 2 bytes) to the stream. This is a utility function which simply calls the `Write` (468) function. The word can be read from the stream using the `ReadWord` (473) function.

Errors: If an error occurs when attempting to write, an `EStreamError` (321) exception will be raised.

See also: `TStream.Write` (468), `TStream.ReadWord` (473), `TStream.WriteByte` (474), `TStream.WriteDWord` (474), `TStream.WriteAnsiString` (475)

4.71.26 TStream.WriteDWord

Synopsis: Write a DWord to the stream.

Declaration: `procedure WriteDWord(d: Cardinal)`

Visibility: `public`

Description: `WriteDWord` writes the `DWord D` (i.e. 4 bytes) to the stream. This is a utility function which simply calls the `Write` (468) function. The `DWord` can be read from the stream using the `ReadDWord` (473) function.

Errors: If an error occurs when attempting to write, an `EStreamError` (321) exception will be raised.

See also: `TStream.Write` (468), `TStream.ReadDWord` (473), `TStream.WriteByte` (474), `TStream.WriteWord` (474), `TStream.WriteString` (475)

4.71.27 TStream.WriteQWord

Synopsis: Write a `QWord` value to the stream.

Declaration: `procedure WriteQWord(q: QWord)`

Visibility: `public`

Description: `WriteQWord` writes the word `W` (i.e. 8 bytes) to the stream. This is a utility function which simply calls the `Write` (468) function. The word can be read from the stream using the `ReadQWord` (473) function.

Errors: If an error occurs when attempting to write, an `EStreamError` (321) exception will be raised.

See also: `TStream.Write` (468), `TStream.ReadByte` (472), `TStream.WriteWord` (474), `TStream.WriteDWord` (474), `TStream.WriteString` (475)

4.71.28 TStream.WriteString

Synopsis: Write an ansistring to the stream.

Declaration: `procedure WriteAnsiString(const S: string); Virtual`

Visibility: `public`

Description: `WriteAnsiString` writes the `AnsiString S` (i.e. 4 bytes) to the stream. This is a utility function which simply calls the `Write` (468) function. The ansistring is written as a 4 byte length specifier, followed by the ansistring's content. The ansistring can be read from the stream using the `ReadAnsiString` (474) function.

Errors: If an error occurs when attempting to write, an `EStreamError` (321) exception will be raised.

See also: `TStream.Write` (468), `TStream.ReadAnsiString` (474), `TStream.WriteByte` (474), `TStream.WriteWord` (474), `TStream.WriteDWord` (474)

4.71.29 TStream.Position

Synopsis: The current position in the stream.

Declaration: `Property Position : Int64`

Visibility: `public`

Access: `Read, Write`

Description: `Position` can be read to determine the current position in the stream. It can be written to set the (absolute) position in the stream. The position is zero-based, so to set the position at the beginning of the stream, the position must be set to zero.

Remark Not all `TStream` descendants support setting the position in the stream, so this should be used with care.

Errors: Some descendants may raise an `EStreamError` (321) exception if they do not support setting the stream position.

See also: `TStream.Size` (476), `TStream.Seek` (468)

4.71.30 TStream.Size

Synopsis: The current size of the stream.

Declaration: `Property Size : Int64`

Visibility: `public`

Access: `Read,Write`

Description: `Size` can be read to determine the stream size or to set the stream size.

Remark Not all descendants of `TStream` support getting or setting the stream size; they may raise an exception if the `Size` property is read or set.

See also: `TStream.Position` (475), `TStream.Seek` (468)

4.72 TStreamAdapter

4.72.1 Description

Implements `IStream` for `TStream` (466) descendants.

4.72.2 Interfaces overview

Page	Interfaces	Description
2055	<code>IStream</code>	COM stream abstraction.

4.72.3 Method overview

Page	Method	Description
480	<code>Clone</code>	Clone the stream.
479	<code>Commit</code>	Commit data to the stream.
478	<code>CopyTo</code>	Copy data to destination stream.
477	<code>Create</code>	Create a new instance of <code>TStreamAdapter</code> .
477	<code>Destroy</code>	Free the <code>TStreamAdapter</code> instance.
479	<code>LockRegion</code>	Lock a region of the stream.
477	<code>Read</code>	Read from the stream.
479	<code>Revert</code>	Revert operations on the stream.
478	<code>Seek</code>	Set the stream position.
478	<code>SetSize</code>	Set the stream size.
480	<code>Stat</code>	Return statistical data about the stream.
479	<code>UnlockRegion</code>	Unlock a region of the stream.
477	<code>Write</code>	Write to the stream.

4.72.4 Property overview

Page	Properties	Access	Description
480	Stream	r	Stream on which adaptor works.
480	StreamOwnership	rw	Determines what happens with the stream when the adaptor is freed.

4.72.5 TStreamAdapter.Create

Synopsis: Create a new instance of `TStreamAdapter`.

Declaration: `constructor Create(Stream: TStream; Ownership: TStreamOwnership)`

Visibility: `public`

Description: `Create` creates a new instance of `TStreamAdapter`. It initializes `TStreamAdapter.Stream` ([480](#)) with `Stream` and initializes `StreamOwnership` ([480](#)) with `Ownership`.

`TStreamAdapter` is an abstract class: descendants must be created that implement the actual functionality.

See also: `StreamOwnership` ([480](#)), `TStreamAdapter.Stream` ([480](#))

4.72.6 TStreamAdapter.Destroy

Synopsis: Free the `TStreamAdapter` instance.

Declaration: `destructor Destroy; Override`

Visibility: `public`

Description: Explicitly free the `TStreamAdapter` instance. Normally, this is done automatically if a reference to the `IStream` interface is freed.

4.72.7 TStreamAdapter.Read

Synopsis: Read from the stream.

Declaration: `function Read(pv: Pointer; cb: DWORD; pcbRead: PDWord) : HRESULT; Virtual`

Visibility: `public`

Description: `Read` implements `#rtl.types.ISequentialStream.Read` ([2054](#)) by reading from the stream specified at creation.

Errors: This function must be overridden and will raise a `runerror 217` when called directly.

See also: `#rtl.types.ISequentialStream.Read` ([2054](#))

4.72.8 TStreamAdapter.Write

Synopsis: Write to the stream.

Declaration: `function Write(pv: Pointer; cb: DWORD; pcbWritten: PDWord) : HRESULT; Virtual`

Visibility: `public`

Description: `Write` implements `#rtl.types.ISequentialStream.Write` (2054) by writing to the stream specified at creation.

Errors: This function must be overridden and will raise a `runerror 217` when called directly.

See also: `#rtl.types.ISequentialStream.Write` (2054)

4.72.9 TStreamAdapter.Seek

Synopsis: Set the stream position.

Declaration: `function Seek(dlibMove: Largeint; dwOrigin: DWORD;
out libNewPosition: LargeUint) : HRESULT; Virtual`

Visibility: `public`

Description: `Seek` implements `#rtl.types.IStream.Seek` (2055) by setting the position of the stream specified at creation.

Errors: This function must be overridden and will raise a `runerror 217` when called directly.

See also: `#rtl.types.IStream.Seek` (2055)

4.72.10 TStreamAdapter.SetSize

Synopsis: Set the stream size.

Declaration: `function SetSize(libNewSize: LargeUint) : HRESULT; Virtual`

Visibility: `public`

Description: `SetSize` implements `#rtl.types.IStream.Setsize` (2055) by setting the size of the stream specified at creation.

Errors: This function must be overridden and will raise a `runerror 217` when called directly.

See also: `#rtl.types.IStream.Setsize` (2055)

4.72.11 TStreamAdapter.CopyTo

Synopsis: Copy data to destination stream.

Declaration: `function CopyTo(stm: IStream; cb: LargeUint; out cbRead: LargeUint;
out cbWritten: LargeUint) : HRESULT; Virtual`

Visibility: `public`

Description: `CopyTo` implements `#rtl.types.IStream.CopyTo` (2055).

Errors: This function must be overridden and will raise a `runerror 217` when called directly.

4.72.12 TStreamAdapter.Commit

Synopsis: Commit data to the stream.

Declaration: `function Commit(grfCommitFlags: DWORD) : HRESULT; Virtual`

Visibility: public

Description: `Commit` implements `#rtl.types.IStream.Commit` (2056).

Errors: This function must be overridden and will raise a `runerror 217` when called directly.

See also: `#rtl.types.IStream.Commit` (2056)

4.72.13 TStreamAdapter.Revert

Synopsis: Revert operations on the stream.

Declaration: `function Revert : HRESULT; Virtual`

Visibility: public

Description: `Revert` implements `#rtl.types.IStream.Revert` (2056).

Errors: This function must be overridden and will raise a `runerror 217` when called directly.

See also: `#rtl.types.IStream.Revert` (2056)

4.72.14 TStreamAdapter.LockRegion

Synopsis: Lock a region of the stream.

Declaration: `function LockRegion(libOffset: LargeUint; cb: LargeUint;
dwLockType: DWORD) : HRESULT; Virtual`

Visibility: public

Description: `LockRegion` implements `#rtl.types.IStream.LockRegion` (2056).

Errors: This function must be overridden and will raise a `runerror 217` when called directly.

See also: `#rtl.types.IStream.LockRegion` (2056)

4.72.15 TStreamAdapter.UnlockRegion

Synopsis: Unlock a region of the stream.

Declaration: `function UnlockRegion(libOffset: LargeUint; cb: LargeUint;
dwLockType: DWORD) : HRESULT; Virtual`

Visibility: public

Description: `UnLockRegion` implements `#rtl.types.IStream.UnLockRegion` (2056).

Errors: This function must be overridden and will raise a `runerror 217` when called directly.

See also: `#rtl.types.IStream.UnLockRegion` (2056)

4.72.16 TStreamAdapter.Stat

Synopsis: Return statistical data about the stream.

Declaration: `function Stat(out statstg: TStatStg; grfStatFlag: DWORD) : HRESULT
; Virtual`

Visibility: public

Description: `Stat` implements `#rtl.types.IStream.Stat` (2057).

Errors: This function must be overridden and will raise a `runerror 217` when called directly.

See also: `#rtl.types.IStream.Stat` (2057)

4.72.17 TStreamAdapter.Clone

Synopsis: Clone the stream.

Declaration: `function Clone(out stm: IStream) : HRESULT; Virtual`

Visibility: public

Description: `Clone` implements `#rtl.types.IStream.Clone` (2057).

Errors: This function must be overridden and will raise a `runerror 217` when called directly.

See also: `#rtl.types.IStream.Clone` (2057)

4.72.18 TStreamAdapter.Stream

Synopsis: Stream on which adaptor works.

Declaration: `Property Stream : TStream`

Visibility: public

Access: Read

Description: This is the stream on which the adaptor works. It was specified at creation.

4.72.19 TStreamAdapter.StreamOwnership

Synopsis: Determines what happens with the stream when the adaptor is freed.

Declaration: `Property StreamOwnership : TStreamOwnership`

Visibility: public

Access: Read,Write

Description: `StreamOwnership` determines what happens when the adaptor

4.73 TStringList

4.73.1 Description

`TStringList` is a descendant class of `TStrings` (486) that implements all of the abstract methods introduced there. It also introduces some additional methods:

- Sort the list, or keep the list sorted at all times
- Special handling of duplicates in sorted lists
- Notification of changes in the list

See also: `TStrings` (486), `TStringList.Duplicates` (484), `TStringList.Sorted` (485)

4.73.2 Method overview

Page	Method	Description
482	Add	Implements the <code>TStrings.Add</code> (491) function.
482	AddObject	
482	Clear	Implements the <code>TStrings.Clear</code> (494) function.
481	Create	
484	CustomSort	Sort the stringlist using a custom sort algorithm.
482	Delete	Implements the <code>TStrings.Delete</code> (494) function.
482	Destroy	Destroys the stringlist.
483	Exchange	Implements the <code>TStrings.Exchange</code> (495) function.
483	Find	Locates the index for a given string in sorted lists.
483	IndexOf	Overrides the <code>TStrings.IndexOf</code> (497) property.
483	Insert	Overrides the <code>TStrings.Insert</code> (499) method.
484	Sort	Sorts the strings in the list.

4.73.3 Property overview

Page	Properties	Access	Description
485	CaseSensitive	rw	Indicates whether comparing strings happens in a case sensitive manner.
484	Duplicates	rw	Describes the behaviour of a sorted list with respect to duplicate strings.
485	OnChange	rw	Event triggered after the list was modified.
486	OnChanging	rw	Event triggered when the list is about to be modified.
486	OwnsObjects	rw	Determines whether the stringlist owns it's objects or not.
485	Sorted	rw	Determines whether the list is sorted or not.
486	SortStyle	rw	Sort style for strings.

4.73.4 TStringList.Create

Declaration: `constructor Create`
`constructor Create(anOwnsObjects: Boolean)`

Visibility: `public`

4.73.5 TStringList.Destroy

Synopsis: Destroys the stringlist.

Declaration: `destructor Destroy; Override`

Visibility: `public`

Description: `Destroy` clears the stringlist, release all memory allocated for the storage of the strings, and then calls the inherited `destroy` method.

Remark Any objects associated to strings in the list will *not* be destroyed; it is the responsibility of the caller to destroy all objects associated with strings in the list.

4.73.6 TStringList.Add

Synopsis: Implements the `TStrings.Add` (491) function.

Declaration: `function Add(const S: string) : Integer; Override`

Visibility: `public`

Description: `Add` will add `S` to the list. If the list is sorted and the string `S` is already present in the list and `TStringList.Duplicates` (484) is `dupError` then an `EStringListError` (322) exception is raised. If `Duplicates` is set to `dupIgnore` then the return value is the index of the previous entry.

If the list is sorted, new strings will not necessarily be added to the end of the list, rather they will be inserted at their alphabetical position.

Errors: If the list is sorted and the string `S` is already present in the list and `TStringList.Duplicates` (484) is `dupError` then an `EStringListError` (322) exception is raised.

See also: `TStringList.Insert` (483), `TStringList.Duplicates` (484)

4.73.7 TStringList.AddObject

Declaration: `function AddObject(const S: string; AObject: TObject) : Integer; Override`

Visibility: `public`

4.73.8 TStringList.Clear

Synopsis: Implements the `TStrings.Clear` (494) function.

Declaration: `procedure Clear; Override`

Visibility: `public`

Description: Implements the `TStrings.Clear` (494) function.

4.73.9 TStringList.Delete

Synopsis: Implements the `TStrings.Delete` (494) function.

Declaration: `procedure Delete(Index: Integer); Override`

Visibility: `public`

Description: Implements the `TStrings.Delete` (494) function.

4.73.10 TStringList.Exchange

Synopsis: Implements the TStrings.Exchange (495) function.

Declaration: `procedure Exchange(Index1: Integer; Index2: Integer); Override`

Visibility: public

Description: Exchange will exchange two items in the list as described in TStrings.Exchange (495).

Remark Exchange will not check whether the list is sorted or not; if Exchange is called on a sorted list and the strings are not identical, the sort order of the list will be destroyed.

See also: TStringList.Sorted (485), TStrings.Exchange (495)

4.73.11 TStringList.Find

Synopsis: Locates the index for a given string in sorted lists.

Declaration: `function Find(const S: string; out Index: Integer) : Boolean; Virtual`

Visibility: public

Description: Find returns True if the string S is present in the list. Upon exit, the Index parameter will contain the position of the string in the list. If the string is not found, the function will return False and Index will contain the position where the string will be inserted if it is added to the list.

Remark

1. Use this method only on sorted lists. For unsorted lists, use TStringList.IndexOf (483) instead.
2. Find uses a binary search method to locate the string

4.73.12 TStringList.IndexOf

Synopsis: Overrides the TStrings.IndexOf (497) property.

Declaration: `function IndexOf(const S: string) : Integer; Override`

Visibility: public

Description: IndexOf overrides the ancestor method TStrings.indexOf (497). It tries to optimize the search by executing a binary search if the list is sorted. The function returns the position of S if it is found in the list, or -1 if the string is not found in the list.

See also: TStrings.IndexOf (497), TStringList.Find (483)

4.73.13 TStringList.Insert

Synopsis: Overrides the TStrings.Insert (499) method.

Declaration: `procedure Insert(Index: Integer; const S: string); Override`

Visibility: public

Description: Insert will insert the string S at position Index in the list. If the list is sorted, an EStringListError (322) exception will be raised instead. Index is a zero-based position.

Errors: If Index contains an invalid value (less than zero or larger than Count, or the list is sorted, an EStringListError (322) exception will be raised.

See also: TStringList.Add (482), TStrings.Insert (499), TStrings.InsertObject (499)

4.73.14 TStringList.Sort

Synopsis: Sorts the strings in the list.

Declaration: `procedure Sort; Virtual`

Visibility: `public`

Description: `Sort` will sort the strings in the list using the quicksort algorithm. If the list has its `TStringList.Sorted` (485) property set to `True` then nothing will be done.

See also: `TStringList.Sorted` (485)

4.73.15 TStringList.CustomSort

Synopsis: Sort the stringlist using a custom sort algorithm.

Declaration: `procedure CustomSort(CompareFn: TStringListSortCompare); Virtual`

Visibility: `public`

Description: `CustomSort` sorts the stringlist with a custom comparison function. The function should compare 2 elements in the list, and return a negative number if the first item is before the second. It should return 0 if the elements are equal, and a positive result indicates that the second elements should be before the first.

See also: `TStringList.Sorted` (485), `TStringList.Sort` (484)

4.73.16 TStringList.Duplicates

Synopsis: Describes the behaviour of a sorted list with respect to duplicate strings.

Declaration: `Property Duplicates : TDuplicates`

Visibility: `public`

Access: `Read,Write`

Description: `Duplicates` describes what to do in case a duplicate value is added to the list:

Table 4.32:

<code>dupIgnore</code>	Duplicate values will not be added to the list, but no error will be triggered.
<code>dupError</code>	If an attempt is made to add a duplicate value to the list, an <code>EStringListError</code> (322) exception is raised.
<code>dupAccept</code>	Duplicate values can be added to the list.

The default value is `dupAccept`.

If the stringlist is not sorted, the `Duplicates` setting is ignored.

4.73.17 TStringList.Sorted

Synopsis: Determines whether the list is sorted or not.

Declaration: `Property Sorted : Boolean`

Visibility: `public`

Access: `Read,Write`

Description: `Sorted` can be set to `True` in order to cause the list of strings to be sorted. Further additions to the list will be inserted at the correct position so the list remains sorted at all times. Setting the property to `False` has no immediate effect, but will allow strings to be inserted at any position.

Remark

1. When `Sorted` is `True`, `TStringList.Insert` (483) cannot be used. For sorted lists, `TStringList.Add` (482) should be used instead.
2. If `Sorted` is `True`, the `TStringList.Duplicates` (484) setting has effect. This setting is ignored when `Sorted` is `False`.

See also: `TStringList.Sort` (484), `TStringList.Duplicates` (484), `TStringList.Add` (482), `TstringList.Insert` (483)

4.73.18 TStringList.CaseSensitive

Synopsis: Indicates whether comparing strings happens in a case sensitive manner.

Declaration: `Property CaseSensitive : Boolean`

Visibility: `public`

Access: `Read,Write`

Description: `CaseSensitive` indicates how string values are compared: When `True` this happens case sensitively, and when `False`, the comparison is done in a case insensitive manner.

This property influences `IndexOf` (497), `Find` (483), `IndexOfName` (498) and `Sort` (484).

See also: `IndexOf` (497), `Find` (483), `IndexOfName` (498), `Sort` (484)

4.73.19 TStringList.OnChange

Synopsis: Event triggered after the list was modified.

Declaration: `Property OnChange : TNotifyEvent`

Visibility: `public`

Access: `Read,Write`

Description: `OnChange` can be assigned to respond to changes that have occurred in the list. The handler is called whenever strings are added, moved, modified or deleted from the list.

The `OnChange` event is triggered after the modification took place. When the modification is about to happen, an `TstringList.OnChanging` (486) event occurs.

See also: `TStringList.OnChanging` (486)

4.73.20 TStringList.OnChanging

Synopsis: Event triggered when the list is about to be modified.

Declaration: `Property OnChanging : TNotifyEvent`

Visibility: `public`

Access: `Read,Write`

Description: `OnChanging` can be assigned to respond to changes that will occurred in the list. The handler is called whenever strings will be added, moved, modified or deleted from the list.

The `Onchanging` event is triggered before the modification will take place. When the modification has happened, an `TStringList.OnChange` ([485](#)) event occurs.

See also: `TStringList.OnChange` ([485](#))

4.73.21 TStringList.OwnsObjects

Synopsis: Determines whether the stringlist owns it's objects or not.

Declaration: `Property OwnsObjects : Boolean`

Visibility: `public`

Access: `Read,Write`

Description: `OwnsObjects` can be set to `true` to let the stringlist instance own the objects in the list: if an element is removed from the list, the associated object (if there is any) will be freed as well. The same is true if the list is cleared or destroyed.

See also: `TStrings.Objects` ([508](#))

4.73.22 TStringList.SortStyle

Synopsis: Sort style for strings.

Declaration: `Property SortStyle : TStringsSortStyle`

Visibility: `public`

Access: `Read,Write`

Description: `SortStyle` sets the sort style for the `TStringList` instance. Setting it to `sslAuto` will keep the list sorted automatically at all times.

`Find` ([483](#)) (and hence `IndexOf` ([497](#))) will use a binary search if `SortStyle` differs from `sslNone`. if `SortStyle` is `sslNone`, then it will raise an exception.

See also: `TStringsSortStyle` ([298](#)), `TStrings.Sorted` ([486](#))

4.74 TStrings

4.74.1 Description

`TStrings` implements an abstract class to manage an array of strings. It introduces methods to set and retrieve strings in the array, searching for a particular string, concatenating the strings and so on. It also allows an arbitrary object to be associated with each string.

It also introduces methods to manage a series of `name=value` settings, as found in many configuration files.

An instance of `TStrings` is never created directly, instead a descendant class such as `TStringList` (481) should be created. This is because `TStrings` is an abstract class which does not implement all methods; `TStrings` also doesn't store any strings, this is the functionality introduced in descendants such as `TStringList` (481).

`TStrings` implements the `IFPObserved` (323) interface: when the stringlist is changed, a `ooChanged` notification is sent to all observers.

See also: `TStringList` (481), `IFPObserved` (323)

4.74.2 Method overview

Page	Method	Description
491	Add	Add a string to the list.
492	AddCommaText	Adds a comma-separated list of strings.
493	AddDelimitedtext	Adds a separator-delimited list of strings.
491	AddObject	Add a string and associated object to the list.
491	AddPair	Add a name-value pair.
492	AddStrings	Add contents of another stringlist to this list.
492	AddText	Add text to the string list.
493	Append	Add a string to the list.
493	Assign	Assign the contents of another stringlist to this one.
494	BeginUpdate	Mark the beginning of an update batch.
494	Clear	Removes all strings and associated objects from the list.
497	Contains	
498	ContainsName	
497	ContainsObject	
489	Create	Initializ a new <code>TStrings</code> instance.
494	Delete	Delete a string from the list.
490	Destroy	Frees all strings and objects, and removes the list from memory.
495	EndUpdate	Mark the end of an update batch.
495	Equals	Compares the contents of two stringlists.
495	Exchange	Exchanges two strings in the list.
496	ExtractName	Extract the name part of a string.
496	Fill	Set a range of strings in the list to the specified value.
496	Filter	Return a filtered list.
496	ForEach	Call a callback for each string in the list.
497	GetEnumerator	Create an <code>IEnumerator</code> instance.
497	GetNameValue	Return both name and value of a name,value pair based on it's index.
497	GetText	Returns the contents as a <code>PChar</code> .
497	IndexOf	Find a string in the list and return its position.
498	IndexOfName	Finds the index of a name in the name-value pairs.
498	IndexOfObject	Finds an object in the list and returns its index.
499	Insert	Insert a string in the list.
499	InsertObject	Insert a string and associated object in the list.
499	LastIndexOf	Return the index of the last occurrence of a string.
500	LoadFromFile	Load the contents of a file as a series of strings.
500	LoadFromStream	Load the contents of a stream as a series of strings.
500	Map	Call a map routine for each string in the list.
501	Move	Move a string from one place in the list to another.
501	Pop	Remove the last string of the list and return it.
501	Reduce	Get a reduced value from the string list.
502	Reverse	Reverse the order of the strings in the list.
502	SaveToFile	Save the contents of the list to a file.
502	SaveToStream	Save the contents of the string to a stream.
492	SetStrings	Replace the strings with the passed strings.
503	SetText	Set the contents of the list from a <code>PChar</code> .
503	Shift	Return and remove the first string in the list.
503	Slice	Return a subrange of strings starting at a given index.
490	ToObjectArray	Return the objects as an array of object.
490	ToStringArray	Return the strings as an array of string.

4.74.3 Property overview

Page	Properties	Access	Description
504	AlwaysQuote	rw	Always quote strings in DelimitedText.
504	Capacity	rw	Capacity of the list, i.e. number of strings that the list can currently hold before it tries to expand.
504	CommaText	rw	Contents of the list as a comma-separated string.
505	Count	r	Number of strings in the list.
506	DefaultEncoding	rw	Default encoding of stringlist.
506	DelimitedText	rw	Get or set all strings in the list in a delimited form.
506	Delimiter	rw	Delimiter character used in DelimitedText (506).
507	Encoding	r	Current encoding of stringlist.
507	LineBreak	rw	LineBreak character to use.
507	MissingNameValueSeparatorAction	rw	What to do if NameValueSeparator is missing in a string.
507	Names	r	Name parts of the name-value pairs in the list.
508	NameValueSeparator	rw	Value of the character used to separate name,value pairs.
508	Objects	rw	Indexed access to the objects associated with the strings in the list.
508	Options	rw	A set of TStringsOption (298) - various boolean properties.
509	QuoteChar	rw	Quote character used in DelimitedText (506).
509	SkipLastLineBreak	rw	Do not add a linebreak to the last item.
509	StrictDelimiter	rw	Should only the delimiter character be considered a delimiter.
510	Strings	rw	Indexed access to the strings in the list.
510	StringsAdapter	rw	Not implemented in Free Pascal.
510	Text	rw	Contents of the list as one big string.
511	TextLineBreakStyle	rw	Determines which line breaks to use in the Text (510) property.
509	TrailingLineBreak	rw	Add a linebreak to the last item.
511	UseLocale	rw	Determines what methods are used in strings comparison.
512	ValueFromIndex	rw	Return the value part of a string based on it's index.
512	Values	rw	Value parts of the name-value pairs in the list.
512	WriteBOM	rw	Write BOM when writing stringlist to stream.

4.74.4 TStrings.Create

Synopsis: Initializ a new TStrings instance.

Declaration: constructor Create

Visibility: public

Description: `Create` initializes the codepage variables of a new `TStrings` instance. Note that `TStrings` is an abstract class, you must always instantiate a descendent such as `TStringList` ([481](#))

Errors: Only an `EOutOfMemory` exception can occur.

See also: `TStringList` ([481](#))

4.74.5 TStrings.Destroy

Synopsis: Frees all strings and objects, and removes the list from memory.

Declaration: `destructor Destroy; Override`

Visibility: `public`

Description: `Destroy` is the destructor of `TStrings` it does nothing except calling the inherited destructor.

4.74.6 TStrings.ToArray

Synopsis: Return the objects as an array of object.

Declaration: `function ToArray(aStart: Integer; aEnd: Integer) : TObjectDynArray
; Overload
function ToArray : TObjectDynArray; Overload`

Visibility: `public`

Description: `ToArray` returns all the values in the `Objects` ([508](#)) array as a dynamic array of `TObject` instances. If `aStart` and `aEnd` are specified, only the objects with indexes in the specified array (`aStart` and `aEnd` included) are returned. `Nil` values will also be returned.

Errors: If `aStart` is larger than `aEnd`, no elements are returned.

See also: `TStrings.ToArray` ([490](#)), `TStrings.Strings` ([510](#)), `TStrings.Objects` ([508](#))

4.74.7 TStrings.ToStringArray

Synopsis: Return the strings as an array of string.

Declaration: `function ToStringArray(aStart: Integer; aEnd: Integer) : TStringDynArray
; Overload
function ToStringArray : TStringDynArray; Overload`

Visibility: `public`

Description: `ToStringArray` returns all the values in the `Strings` ([510](#)) array as a dynamic array of `TString` instances. If `aStart` and `aEnd` are specified, only the strings with indexes in the specified array (`aStart` and `aEnd` included) are returned. `Nil` values will also be returned.

Errors: If `aStart` is larger than `aEnd`, no elements are returned.

See also: `TStrings.ToArray` ([490](#)), `TStrings.Strings` ([510](#)), `TStrings.Objects` ([508](#))

4.74.8 TStrings.Add

Synopsis: Add a string to the list.

Declaration: `function Add(const S: string) : Integer; Virtual; Overload`
`function Add(const Fmt: string; const Args: Array of const) : Integer`
`; Overload`

Visibility: public

Description: Add adds S at the end of the list and returns the index of S in the list (which should equal TStrings.Count (505))

See also: TStrings.Objects (508), TStrings.AddObject (491), TStrings.Insert (499), TStrings.Delete (494), TStrings.Strings (510), TStrings.Count (505)

4.74.9 TStrings.AddObject

Synopsis: Add a string and associated object to the list.

Declaration: `function AddObject(const S: string; AObject: TObject) : Integer`
`; Virtual; Overload`
`function AddObject(const Fmt: string; Args: Array of const;`
`AObject: TObject) : Integer; Overload`

Visibility: public

Description: AddObject adds S to the list of strings, and associates AObject with it. It returns the index of S.

Remark An object added to the list is not automatically destroyed by the list when the list is destroyed or the string it is associated with is deleted. It is the responsibility of the application to destroy any objects associated with strings.

See also: TStrings.Add (491), TStrings.Strings (510), TStrings.Objects (508), TStrings.InsertObject (499)

4.74.10 TStrings.AddPair

Synopsis: Add a name-value pair.

Declaration: `function AddPair(const AName: string; const AValue: string) : TStrings`
`; Overload`
`function AddPair(const AName: string; const AValue: string;`
`AObject: TObject) : TStrings; Overload`

Visibility: public

Description: AddPair adds a Name=Value pair with the AName and AValue parameters, optionally with object AObject. It uses the correct NameValueSeparator (281) character to do so.

AddPair does not test whether aName already exists, so it is possible to add duplicate names.

Errors: None.

See also: TStrings.GetNameValue (497), TStrings.NameValueSeparator (508), TStrings.Add (491), TStrings.AddObject (491)

4.74.11 TStrings.AddStrings

Synopsis: Add contents of another stringlist to this list.

```

Declaration: procedure AddStrings(TheStrings: TStrings); Virtual; Overload
                procedure AddStrings(TheStrings: TStrings; ClearFirst: Boolean)
                        ; Overload
                procedure AddStrings(const TheStrings: Array of string); Virtual
                        ; Overload
                procedure AddStrings(const TheStrings: Array of string;
                        ClearFirst: Boolean); Overload

```

Visibility: public

Description: `AddStrings` adds the contents of `TheStrings` to the stringlist. Any associated objects are added as well.

See also: TStrings.Add (491), TStrings.Assign (493)

4.74.12 TStrings.SetStrings

Synopsis: Replace the strings with the passed strings.

```
Declaration: procedure SetStrings(TheStrings: TStrings); Virtual; Overload
           procedure SetStrings(TheStrings: Array of string); Virtual; Overload
```

Visibility: public

Description: `SetStrings` sets the strings of the stringlist to the specified values in `TheStrings`. The values can be specified as another stringlist or as an array of strings. In difference with the `Assign` (493) method it does not copy all other properties.

Internally it clears the array and calls `TStrings.AddStrings` (492).

See also: `TStrings.Assign` (493), `TStrings.AddStrings` (492)

4.74.13 TStrings.AddText

Synopsis: Add text to the string list.

```
Declaration: procedure AddText(const S: string); Virtual
```

Visibility: public

Description: `AddText` adds `S` to the strings. It is identical in function to setting `Text` (281) but does not clear the list of strings first: `S` is split into lines, and each line is added to the list.

See also: [TString.Text \(281\)](#)

4.74.14 TStrings.AddCommaText

Synopsis: Adds a comma-separated list of strings.

```
Declaration: procedure AddCommaText (const S: string)
```

Visibility: public

Description: `AddCommaText` parses `S` using the same mechanism as when setting `TStrings.CommaText` (504) but adds the values instead of removing existing values.

`AddCommaText` also observes the `TStrings.StrictDelimiter` (509) and `TStrings.QuoteChar` (509) properties.

See also: `TStrings.AddDelimitedText` (493), `TStrings.CommaText` (504), `TStrings.QuoteChar` (509), `TStrings.StrictDelimiter` (509)

4.74.15 TStrings.AddDelimitedtext

Synopsis: Adds a separator-delimited list of strings.

Declaration: `procedure AddDelimitedText(const S: string; ADelimiter: Char;`
`AStrictDelimiter: Boolean); Overload`
`procedure AddDelimitedtext(const S: string); Overload`

Visibility: `public`

Description: `AddDelimitedText` parses `S` using the same mechanism as when setting `TStrings.DelimitedText` (506) but adds the values instead of removing existing values. In difference with `TStrings.AddCommaText` (492) it uses the `TStrings.Delimiter` (506) character to separate the values.

`AddDelimitedText` also observes the `TStrings.Delimiter` (506), `TStrings.StrictDelimiter` (509) and `TStrings.QuoteChar` (509) properties.

See also: `TStrings.AddCommaText` (492), `TStrings.DelimitedText` (506), `TStrings.Delimiter` (506), `TStrings.StrictDelimiter` (509), `TStrings.QuoteChar` (509)

4.74.16 TStrings.Append

Synopsis: Add a string to the list.

Declaration: `procedure Append(const S: string)`

Visibility: `public`

Description: `Append` does the same as `TStrings.Add` (491), only it does not return the index of the inserted string.

See also: `TStrings.Add` (491)

4.74.17 TStrings.Assign

Synopsis: Assign the contents of another stringlist to this one.

Declaration: `procedure Assign(Source: TPersistent); Override`

Visibility: `public`

Description: `Assign` replaces the contents of the stringlist with the contents of `Source` if `Source` is also of type `TStrings`. Any associated objects are copied as well.

See also: `TStrings.Add` (491), `TStrings.AddStrings` (492), `TPersistent.Assign` (447)

4.74.18 TStrings.BeginUpdate

Synopsis: Mark the beginning of an update batch.

Declaration: `procedure BeginUpdate`

Visibility: `public`

Description: `BeginUpdate` increases the update count by one. It is advisable to call `BeginUpdate` before lengthy operations on the stringlist. At the end of these operation, `TStrings.EndUpdate` (495) should be called to mark the end of the operation. Descendant classes may use this information to perform optimizations. e.g. updating the screen only once after many strings were added to the list.

All `TStrings` methods that modify the string list call `BeginUpdate` before the actual operation, and call `endUpdate` when the operation is finished. Descendant classes should also call these methods when modifying the string list.

Remark Always put the corresponding call to `TStrings.EndUpdate` (495) in the context of a `Finally` block, to ensure that the update count is always decreased at the end of the operation, even if an exception occurred:

```
With MyStrings do
  try
    BeginUpdate;
    // Some lengthy operation.
  Finally
    EndUpdate
  end;
```

See also: `TStrings.EndUpdate` (495)

4.74.19 TStrings.Clear

Synopsis: Removes all strings and associated objects from the list.

Declaration: `procedure Clear; Virtual; Abstract`

Visibility: `public`

Description: `Clear` will remove all strings and their associated objects from the list. After a call to `clear`, `TStrings.Count` (505) is zero.

Since it is an abstract method, `TStrings` itself does not implement `Clear`. Descendant classes such as `TStringList` (481) implement this method.

See also: `TStrings.Objects` (508), `TStrings.Strings` (510), `TStrings.Delete` (494), `TStrings.Count` (505)

4.74.20 TStrings.Delete

Synopsis: Delete a string from the list.

Declaration: `procedure Delete(Index: Integer); Virtual; Abstract`

Visibility: `public`

Description: `Delete` deletes the string at position `Index` from the list. The associated object is also removed from the list, but not destroyed. `Index` is zero-based, and should be in the range 0 to `Count-1`.

Since it is an abstract method, `TStrings` itself does not implement `Delete`. Descendant classes such as `TStringList` (481) implement this method.

Errors: If `Index` is not in the allowed range, an `EStringListError` (322) is raised.

See also: `TStrings.Insert` (499), `TStrings.Objects` (508), `TStrings.Strings` (510), `TStrings.Clear` (494)

4.74.21 TStrings.EndUpdate

Synopsis: Mark the end of an update batch.

Declaration: `procedure EndUpdate`

Visibility: `public`

Description: `EndUpdate` should be called at the end of a lengthy operation on the stringlist, but only if there was a call to `BeginUpdate` before the operation was started. It is best to put the call to `EndUpdate` in the context of a `Finally` block, so it will be called even if an exception occurs.

For more information, see `TStrings.BeginUpdate` (494).

`TStrings` implements the `IFPObserved` (323) interface: when `EndUpdate` is called, a `ooChanged` notification is sent to all observers.

See also: `TStrings.BeginUpdate` (494), `IFPObserved` (323)

4.74.22 TStrings.Equals

Synopsis: Compares the contents of two stringlists.

Declaration: `function Equals(Obj: TObject) : Boolean; Override; Overload`
`function Equals(TheStrings: TStrings) : Boolean; Overload`

Visibility: `public`

Description: `Equals` compares the contents of the stringlist with the contents of `TheStrings`. If the contents match, i.e. the stringlist contain an equal amount of strings, and all strings match, then `True` is returned. If the number of strings in the lists is unequal, or they contain one or more different strings, `False` is returned.

Remark

- 1.The strings are compared case-insensitively.
- 2.The associated objects are not compared

See also: `TStrings.Objects` (508), `TStrings.Strings` (510), `TStrings.Count` (505), `TStrings.Assign` (493)

4.74.23 TStrings.Exchange

Synopsis: Exchanges two strings in the list.

Declaration: `procedure Exchange(Index1: Integer; Index2: Integer); Virtual`

Visibility: `public`

Description: `Exchange` exchanges the strings at positions `Index1` and `Index2`. The associated objects are also exchanged.

Both indexes must be in the range of valid indexes, i.e. must have a value between 0 and `Count-1`.

Errors: If either `Index1` or `Index2` is not in the range of valid indexes, an `EStringListError` (322) exception is raised.

See also: `TStrings.Move` (501), `TStrings.Strings` (510), `TStrings.Count` (505)

4.74.24 TStrings.ExtractName

Synopsis: Extract the name part of a string.

Declaration: `function ExtractName(const S: string) : string`

Visibility: public

Description: `ExtractName` returns the name part (the part before the `NameValueSeparator` (508) character) of the string. If the character is not present, an empty string is returned. The resulting string is not trimmed, it can end or start with spaces.

See also: `NameValueSeparator` (508)

4.74.25 TStrings.Filter

Synopsis: Return a filtered list.

Declaration: `procedure Filter(aFilter: TStringsFilterMethod; aList: TStrings)`
`function Filter(aFilter: TStringsFilterMethod) : TStrings`

Visibility: public

Description: `Filter` applies `aFilter` to each string in the list, and if `aFilter` returns `True`, adds the string to the `aList` list. If the `aList` argument is not supplied, the result is a `TStringList` (481) instance that must be freed by the caller.

See also: `TStringsFilterMethod` (297), `TStringList` (481), `TStrings.Foreach` (496), `TStrings.Map` (500), `TStrings.Reduce` (501), `TStrings.Reverse` (502)

4.74.26 TStrings.Fill

Synopsis: Set a range of strings in the list to the specified value.

Declaration: `procedure Fill(const aValue: string; aStart: Integer; aEnd: Integer)`

Visibility: public

Description: `Fill` sets the strings with indexes in the range `aStart..aEnd` to `aValue`, if the `aStart..aEnd` are outside the valid range of indexes for the list, their values are adapted to fit the range. (0 to `Count` (505)-1)

See also: `Count` (505), `Strings` (510)

4.74.27 TStrings.Foreach

Synopsis: Call a callback for each string in the list.

Declaration: `procedure Foreach(aCallback: TStringsForeachMethod)`
`procedure Foreach(aCallback: TStringsForeachMethodEx)`
`procedure Foreach(aCallback: TStringsForeachMethodExObj)`

Visibility: public

Description: `Foreach` calls `aCallback` for every string in the list. It passes the string and optionally the string index and associated object to the callback.

See also: `TStrings.Filter` (496), `TStrings.Map` (500), `TStrings.Fill` (496), `TStringsForeachMethod` (297), `TStringsForeachMethodEx` (297), `TStringsForeachMethodExObj` (297), `TStrings.Reverse` (502)

4.74.28 TStrings.GetEnumerator

Synopsis: Create an `IEnumerator` instance.

Declaration: `function GetEnumerator : TStringsEnumerator`

Visibility: public

Description: `GetEnumerator` is the implementation of the `IEnumerable` (1591) interface for `TStrings`. It creates a `TStringsEnumerator` (513) instance and returns it's `IEnumerator` (1591) interface.

See also: `TStringsEnumerator` (513), `IEnumerator` (1591), `IEnumerable` (1591)

4.74.29 TStrings.GetNameValue

Synopsis: Return both name and value of a name,value pair based on it's index.

Declaration: `procedure GetNameValue(Index: Integer; out AName: string;
out AValue: string)`

Visibility: public

Description: Return both name and value of a name,value pair based on it's index.

4.74.30 TStrings.GetText

Synopsis: Returns the contents as a `PChar`.

Declaration: `function GetText : PChar; Virtual`

Visibility: public

Description: `GetText` allocates a memory buffer and copies the contents of the stringlist to this buffer as a series of strings, separated by an end-of-line marker. The buffer is zero terminated.

Remark The caller is responsible for freeing the returned memory buffer.

4.74.31 TStrings.Contains

Declaration: `function Contains(const S: string) : Boolean`

Visibility: public

4.74.32 TStrings.ContainsObject

Declaration: `function ContainsObject(const AObject: TObject) : Boolean`

Visibility: public

4.74.33 TStrings.IndexOf

Synopsis: Find a string in the list and return its position.

Declaration: `function IndexOf(const S: string) : Integer; Virtual
function IndexOf(const S: string; aStart: Integer) : Integer; Virtual`

Visibility: public

Description: `IndexOf` searches the list for `S` starting from index `aStart`. The start position `aStart` is 0 if not specified. If it is negative, it is considered an offset from `TStrings.Count` (505). The search is case-insensitive or case sensitive depending on the value of the `CaseSensitive` (??) . If a matching entry is found, its position is returned. if no matching string is found, `-1` is returned.

Remark

1. Only the first occurrence of the string is returned.
2. The returned position is zero-based, i.e. 0 indicates the first string in the list.

See also: `TStrings.IndexOfObject` (498), `TStrings.IndexOfName` (498), `TStrings.Strings` (510), `TStrings.LastIndexOf` (499), `TStrings.Count` (505)

4.74.34 TStrings.ContainsName

Declaration: `function ContainsName(const Name: string) : Boolean; Virtual`

Visibility: public

4.74.35 TStrings.IndexOfName

Synopsis: Finds the index of a name in the name-value pairs.

Declaration: `function IndexOfName(const Name: string) : Integer; Virtual`

Visibility: public

Description: `IndexOfName` searches in the list of strings for a name-value pair with name part `Name`. If such a pair is found, it returns the index of the pair in the stringlist. If no such pair is found, the function returns `-1`. The search is done case-insensitive.

Remark

1. Only the first occurrence of a matching name-value pair is returned.
2. The returned position is zero-based, i.e. 0 indicates the first string in the list.

See also: `TStrings.IndexOf` (497), `TStrings.IndexOfObject` (498), `TStrings.Strings` (510), `TStrings.LastIndexOf` (499)

4.74.36 TStrings.IndexOfObject

Synopsis: Finds an object in the list and returns its index.

Declaration: `function IndexOfObject(AObject: TObject) : Integer; Virtual`

Visibility: public

Description: `IndexOfObject` searches through the list of strings till it find a string associated with `AObject`, and returns the index of this string. If no such string is found, `-1` is returned.

Remark

1. Only the first occurrence of a string with associated object `AObject` is returned; if more strings in the list can be associated with `AObject`, they will not be found by this routine.
2. The returned position is zero-based, i.e. 0 indicates the first string in the list.

See also: `TStrings.LastIndexOf` (499), `TStrings.IndexOf` (497), `TStrings.Objects` (508)

4.74.37 TStrings.Insert

Synopsis: Insert a string in the list.

Declaration: `procedure Insert(Index: Integer; const S: string); Virtual; Abstract`

Visibility: public

Description: `Insert` inserts the string `S` at position `Index` in the list. `Index` is a zero-based position, and can have values from 0 to `Count`. If `Index` equals `Count` then the string is appended to the list.

Remark

1. All methods that add strings to the list use `Insert` to add a string to the list.
2. If the string has an associated object, use `TStrings.InsertObject` (499) instead.

Errors: If `Index` is less than zero or larger than `Count` then an `EStringListError` (322) exception is raised.

See also: `TStrings.Add` (491), `TStrings.InsertObject` (499), `TStrings.Append` (493), `TStrings.Delete` (494)

4.74.38 TStrings.InsertObject

Synopsis: Insert a string and associated object in the list.

Declaration: `procedure InsertObject(Index: Integer; const S: string;
AObject: TObject); Virtual`

Visibility: public

Description: `InsertObject` inserts the string `S` and its associated object `AObject` at position `Index` in the list. `Index` is a zero-based position, and can have values from 0 to `Count`. If `Index` equals `Count` then the string is appended to the list.

Errors: If `Index` is less than zero or larger than `Count` then an `EStringListError` (322) exception is raised.

See also: `TStrings.Insert` (499), `TStrings.AddObject` (491), `TStrings.Append` (493), `TStrings.Delete` (494)

4.74.39 TStrings.LastIndexOf

Synopsis: Return the index of the last occurrence of a string.

Declaration: `function LastIndexOf(const S: string; aStart: Integer) : Integer
; Virtual
function LastIndexOf(const S: string) : Integer`

Visibility: public

Description: `LastIndexOf` returns the index of the last occurrence of the string `S`. If `aStart` is specified, the search starts at the specified index in the list (it defaults to `Count-1`). If `aStart` is negative, it is used as an offset to `Count` (505), i.e. `aCount` equal to -1 will start the search at `Count-1`.

If `aStart` is larger than `Count-1` it will be capped to `Count-1`.

See also: `IndexOf` (497), `IndexOfObject` (498)

4.74.40 TStrings.LoadFromFile

Synopsis: Load the contents of a file as a series of strings.

Declaration: `procedure LoadFromFile(const FileName: string); Virtual; Overload`
`procedure LoadFromFile(const FileName: string; IgnoreEncoding: Boolean)`
`procedure LoadFromFile(const FileName: string; AEncoding: TEncoding)`
`; Virtual; Overload`

Visibility: public

Description: `LoadFromFile` loads the contents of a file into the stringlist. Each line in the file (as marked by the end-of-line marker of the particular OS the application runs on) becomes one string in the stringlist. This action replaces the contents of the stringlist, it does not append the strings to the current content.

`LoadFromFile` simply creates a file stream (406) with the given filename, and then executes `TStrings.LoadfromStream` (500); after that the file stream object is destroyed again.

See also: `TStrings.LoadFromStream` (500), `TStrings.SaveToFile` (502), `Tstrings.SaveToStream` (502)

4.74.41 TStrings.LoadFromStream

Synopsis: Load the contents of a stream as a series of strings.

Declaration: `procedure LoadFromStream(Stream: TStream); Virtual; Overload`
`procedure LoadFromStream(Stream: TStream; IgnoreEncoding: Boolean)`
`; Overload`
`procedure LoadFromStream(Stream: TStream; AEncoding: TEncoding)`
`; Virtual; Overload`

Visibility: public

Description: `LoadFromStream` loads the contents of `Stream` into the stringlist. Each line in the stream (as marked by the end-of-line marker of the particular OS the application runs on) becomes one string in the stringlist. This action replaces the contents of the stringlist, it does not append the strings to the current content.

See also: `TStrings.LoadFromFile` (500), `TStrings.SaveToFile` (502), `Tstrings.SaveToStream` (502)

4.74.42 TStrings.Map

Synopsis: Call a map routine for each string in the list.

Declaration: `procedure Map(aMap: TStringsMapMethod; aList: TStrings)`
`function Map(aMap: TStringsMapMethod) : TStrings`

Visibility: public

Description: `Map` applies `aMap` to each string in the list and adds the result to the list `aList`. If `aList` is not specified, the result is a `TStringList` (481) instance which must be freed by the caller.

See also: `TStrings.ForEach` (496), `TStrings.Map` (500), `TStrings.Filter` (496), `TStringsMapMethod` (297), `TStrings.Reverse` (502)

4.74.43 TStrings.Move

Synopsis: Move a string from one place in the list to another.

Declaration: `procedure Move (CurIndex: Integer; NewIndex: Integer); Virtual`

Visibility: public

Description: `Move` moves the string at position `CurIndex` so it has position `NewIndex` after the move operation. The object associated to the string is also moved. `CurIndex` and `NewIndex` should be in the range of 0 to `Count-1`.

Remark `NewIndex` is *not* the position in the stringlist before the move operation starts. The move operation

- 1.removes the string from position `CurIndex`
- 2.inserts the string at position `NewIndex`

This may not lead to the desired result if `NewIndex` is bigger than `CurIndex`. Consider the following example:

```
With MyStrings do
begin
  Clear;
  Add('String 0');
  Add('String 1');
  Add('String 2');
  Add('String 3');
  Add('String 4');
  Move(1,3);
end;
```

After the `Move` operation has completed, 'String 1' will be between 'String 3' and 'String 4'.

Errors: If either `CurIndex` or `NewIndex` is outside the allowed range, an `EStringListError` (322) is raised.

See also: `TStrings.Exchange` (495)

4.74.44 TStrings.Pop

Synopsis: Remove the last string of the list and return it.

Declaration: `function Pop : string`

Visibility: public

Description: `Pop` returns the last list in the string and removes it from the list. If there are no strings (i.e. `Count` (505) equals zero), an empty string is returned.

See also: `TStrings.Add` (491), `TStrings.Shift` (503), `TStrings.Slice` (503)

4.74.45 TStrings.Reduce

Synopsis: Get a reduced value from the string list.

Declaration: `function Reduce (aReduceMethod: TStringsReduceMethod;
 const startingValue: string) : string`

Visibility: public

Description: `Reduce` iterates over all strings in the list and calls the `aReduceMethod` callback with 2 strings: the result of the previous iteration and the current string. The return value will be used in the next iteration, or will result in the end result of the method if the last string was reached.

See also: `TStrings.Filter` (496), `TStrings.Map` (500), `TStrings.Foreach` (496), `TStrings.ReduceMethod` (298), `TStrings.Reverse` (502)

4.74.46 TStrings.Reverse

Synopsis: Reverse the order of the strings in the list.

Declaration: `function Reverse : TStrings`
`procedure Reverse(aList: TStrings)`

Visibility: public

Description: `Reverse` reverses the order of the strings in the list.

See also: `TStrings.Filter` (496), `TStrings.Map` (500), `TStrings.Foreach` (496), `TStrings.Reduce` (501), `TStrings.Sort` (486)

4.74.47 TStrings.SaveToFile

Synopsis: Save the contents of the list to a file.

Declaration: `procedure SaveToFile(const FileName: string); Virtual; Overload`
`procedure SaveToFile(const FileName: string; IgnoreEncoding: Boolean)`
`; Overload`
`procedure SaveToFile(const FileName: string; AEncoding: TEncoding)`
`; Virtual; Overload`

Visibility: public

Description: `SaveToFile` saves the contents of the stringlist to the file with name `FileName`. It writes the strings to the file, separated by end-of-line markers, so each line in the file will contain 1 string from the stringlist.

`SaveToFile` creates a file stream (406) with name `FileName`, calls `TStrings.SaveToStream` (502) and then destroys the file stream object.

Errors: An `EStreamError` (321) exception can be raised if the file `FileName` cannot be opened, or if it cannot be written to.

See also: `TStrings.SaveToStream` (502), `Tstrings.LoadFromStream` (500), `TStrings.LoadFromFile` (500)

4.74.48 TStrings.SaveToStream

Synopsis: Save the contents of the string to a stream.

Declaration: `procedure SaveToStream(Stream: TStream); Virtual; Overload`
`procedure SaveToStream(Stream: TStream; IgnoreEncoding: Boolean)`
`; Overload`
`procedure SaveToStream(Stream: TStream; AEncoding: TEncoding); Virtual`
`; Overload`

Visibility: public

Description: `SaveToStream` saves the contents of the stringlist to `Stream`. It writes the strings to the stream, separated by end-of-line markers, so each 'line' in the stream will contain 1 string from the stringlist.

Errors: An `EStreamError` (321) exception can be raised if the stream cannot be written to.

See also: `TStrings.SaveToFile` (502), `Tstrings.LoadFromStream` (500), `TStrings.LoadFromFile` (500)

4.74.49 TStrings.Shift

Synopsis: Return and remove the first string in the list.

Declaration: `function Shift : string`

Visibility: public

Description: `Shift` returns the first string in the list and removes it from the list, causing all values to shift one position. If there are no strings in the list, an empty value is returned.

See also: `TStrings.Slice` (503), `TStrings.Pop` (501), `TStrings.Delete` (494), `TStrings.Insert` (499)

4.74.50 TStrings.Slice

Synopsis: Return a subrange of strings starting at a given index.

Declaration:

```
procedure Slice(fromIndex: Integer; aList: TStrings); Overload
procedure Slice(fromIndex: Integer; toIndex: Integer; aList: TStrings)
           ; Overload
function Slice(fromIndex: Integer) : TStrings; Overload
function Slice(fromIndex: Integer; toIndex: Integer) : TStrings
           ; Overload
```

Visibility: public

Description: `Slice` adds to `aList` the strings in the current list, starting with the string at index `FromIndex`. If `aList` is not specified, a `TStringList` (481) instance is created and used. The caller must free the new instance.

See also: `TStrings.Filter` (496), `TStrings.Shift` (503), `TStrings.Pop` (501), `TStrings.Map` (500), `TStrings.Delete` (494), `TStrings.Insert` (499)

4.74.51 TStrings.SetText

Synopsis: Set the contents of the list from a `PChar`.

Declaration: `procedure SetText(TheText: PChar); Virtual`

Visibility: public

Description: `SetText` parses the contents of `TheText` and fills the stringlist based on the contents. It regards `TheText` as a series of strings, separated by end-of-line markers. Each of these strings is added to the stringlist.

See also: `TStrings.Text` (510)

4.74.52 TStrings.AlwaysQuote

Synopsis: Always quote strings in DelimitedText.

Declaration: `Property AlwaysQuote : Boolean`

Visibility: `public`

Access: `Read,Write`

Description: `AlwaysQuote` tells the stringlist instance to quote strings in `DelimitedText` (506) . The default is to quote strings only when they have whitespace in them.

See also: `DelimitedText` (506), `CommaText` (506), `StrictDelimiter` (509)

4.74.53 TStrings.Capacity

Synopsis: Capacity of the list, i.e. number of strings that the list can currently hold before it tries to expand.

Declaration: `Property Capacity : Integer`

Visibility: `public`

Access: `Read,Write`

Description: `Capacity` is the number of strings that the list can hold before it tries to allocate more memory.

`TStrings` returns `TStrings.Count` (505) when read. Trying to set the capacity has no effect. Descendant classes such as `TStringList` (481) can override this property such that it actually sets the new capacity.

See also: `TStringList` (481), `TStrings.Count` (505)

4.74.54 TStrings.CommaText

Synopsis: Contents of the list as a comma-separated string.

Declaration: `Property CommaText : string`

Visibility: `public`

Access: `Read,Write`

Description: `CommaText` represents the stringlist as a single string, consisting of a comma-separated concatenation of the strings in the list. If one of the strings contains spaces, comma's or quotes it will be enclosed by double quotes. Any double quotes in a string will be doubled. For instance the following strings:

```
Comma,string
Quote"string
Space string
NormalString
```

is converted to

```
"Comma,string","Quote""string","Space string",NormalString
```

Conversely, when setting the `CommaText` property, the text will be parsed according to the rules outlined above, and the strings will be set accordingly. Note that spaces will in this context be regarded as string separators, unless the string as a whole is contained in double quotes. If the `TStrings.StrictDelimiter` (509) property is false, then whitespace characters near delimiters will be ignored. The following string:

```
"Comma,string" , "Quote"string",Space string,, NormalString
```

Will be converted to

```
Comma,string
Quote"string
Space
string
```

```
NormalString
```

If the `TStrings.StrictDelimiter` property is true, then the result will be as follows (where spaces are replaced with `_` for clarity):

```
Comma,string
_
_"Quote"string"
Space_string
_NormalString
```

The `CommaText` is a special case of the `TStrings.DelimitedText` (506) property where the quote character is always the double quote (`TStrings QuoteChar` (509) is ignored), and the delimiter is always a comma.

Setting `CommaText` is equivalent to clearing the stringlist and calling `TStrings.AddCommaText` (492).

See also: `TStrings.StrictDelimiter` (509), `TStrings.Text` (510), `TStrings.SetText` (503), `TStrings.DelimitedText` (506), `TStrings.AddCommaText` (492)

4.74.55 TStrings.Count

Synopsis: Number of strings in the list.

Declaration: `Property Count : Integer`

Visibility: `public`

Access: `Read`

Description: `Count` is the current number of strings in the list. `TStrings` does not implement this property; descendant classes should override the property read handler to return the correct value.

Strings in the list are always uniquely identified by their `Index`; the index of a string is zero-based, i.e. it's supported range is 0 to `Count-1`. trying to access a string with an index larger than or equal to `Count` will result in an error. Code that iterates over the list in a stringlist should always take into account the zero-based character of the list index.

See also: `TStrings.Strings` (510), `TStrings.Objects` (508), `TStrings.Capacity` (504)

4.74.56 TStrings.DefaultEncoding

Synopsis: Default encoding of stringlist.

Declaration: `Property DefaultEncoding : TEncoding`

Visibility: `public`

Access: `Read,Write`

Description: `DefaultEncoding` is the default encoding used by the `TStrings` instance. It is not the actual encoding, as specified by `Encoding` (507).

See also: `Encoding` (507)

4.74.57 TStrings.DelimitedText

Synopsis: Get or set all strings in the list in a delimited form.

Declaration: `Property DelimitedText : string`

Visibility: `public`

Access: `Read,Write`

Description: `DelimitedText` returns all strings, properly quoted with `QuoteChar` (509) and separated by the `Delimiter` (506) character.

Strings are quoted if they contain a space or any character with ASCII value less than 32.

The `CommaText` (504) property is a special case of delimited text where the delimiter character is a comma and the quote character is a double quote.

If `StrictDelimiter` (509) is set to `True`, then no quoting is done (The `QuoteChar` property is disregarded completely): the returned text will contain the items in the stringlist, separated by the `Delimiter` character. When writing the `DelimitedText` property, the text will be split at all occurrences of the `Delimiter` character; however, when reading, the `QuoteChar` property will be taken into account.

Setting `TStrings.AddDelimitedText` (493) is equivalent to clearing the list and calling `TStrings.AddDelimitedText` (493).

See also: `TStrings.Delimiter` (506), `TStrings.StrictDelimiter` (509), `TStrings.Text` (510), `TStrings.QuoteChar` (509), `TStrings.CommaText` (504), `TStrings.AddDelimitedText` (493)

4.74.58 TStrings.Delimiter

Synopsis: Delimiter character used in `DelimitedText` (506).

Declaration: `Property Delimiter : Char`

Visibility: `public`

Access: `Read,Write`

Description: `Delimiter` is the delimiter character used to separate the different strings in the stringlist when they are read or set through the `DelimitedText` (506) property.

See also: `TStrings.DelimitedText` (506)

4.74.59 TStrings.Encoding

Synopsis: Current encoding of stringlist.

Declaration: `Property Encoding : TEncoding`

Visibility: public

Access: Read

Description: `Encoding` is the current encoding used by the `TStrings` instance, and which was specified in the constructor. It is not the default encoding, as specified by `DefaultEncoding` (506).

See also: `DefaultEncoding` (506)

4.74.60 TStrings.LineBreak

Synopsis: `LineBreak` character to use.

Declaration: `Property LineBreak : string`

Visibility: public

Access: Read,Write

Description: `LineBreak` is the character used to separate lines when reading or writing `TStrings.Text` (510), and it is also used when reading from file.

See also: `TStrings.Text` (510)

4.74.61 TStrings.MissingNameValueSeparatorAction

Synopsis: What to do if `NameValueSeparator` is missing in a string.

Declaration: `Property MissingNameValueSeparatorAction : TMissingNameValueSeparatorAction`

Visibility: public

Access: Read,Write

Description: `MissingNameValueSeparatorAction` describes what to do if a given string in the list does not have a `NameValueSeparator` (508) character in it: consider the line a name, a value or an empty line, or raise an error. For a list of possible values and their explanation, see `TMissingNameValueSeparatorAction` (292)

See also: `TMissingNameValueSeparatorAction` (292), `IndexOfName` (281), `Names` (281), `Values` (281), `GetNameValue` (281), `ValueFromIndex` (281)

4.74.62 TStrings.Names

Synopsis: Name parts of the name-value pairs in the list.

Declaration: `Property Names[Index: Integer]: string`

Visibility: public

Access: Read

Description: `Names` provides indexed access to the names of the name-value pairs in the list. It returns the name part of the `Index`-th string in the list.

Remark The index is not an index based on the number of name-value pairs in the list. It is the name part of the name-value pair a string `Index` in the list. If the string at position `Index` is not a name-value pair (i.e. does not contain the equal sign (=)), then an empty name is returned.

See also: `TStrings.Values` (512), `TStrings.IndexOfName` (498)

4.74.63 TStrings.NameValueSeparator

Synopsis: Value of the character used to separate name,value pairs.

Declaration: `Property NameValueSeparator : Char`

Visibility: public

Access: Read,Write

Description: `NameValueSeparator` is the character used to separate name,value pair. By default, this is the equal sign (=), resulting in `Name=Value` pairs.

It can be set to a colon for `Name : Value` pairs.

4.74.64 TStrings.Objects

Synopsis: Indexed access to the objects associated with the strings in the list.

Declaration: `Property Objects[Index: Integer]: TObject`

Visibility: public

Access: Read,Write

Description: `Objects` provides indexed access to the objects associated to the strings in the list. `Index` is a zero-based index and must be in the range of 0 to `Count-1`.

Setting the `objects` property will not free the previously associated object, if there was one. The caller is responsible for freeing the object that was previously associated to the string.

`TStrings` does not implement any storage for objects. Reading the `Objects` property will always return `Nil`. Setting the property will have no effect. It is the responsibility of the descendant classes to provide storage for the associated objects.

Errors: If an `Index` outside the valid range is specified, an `EStringListError` (322) exception will be raised.

See also: `TStrings.Strings` (510), `TStrings.IndexOfObject` (498), `TStrings.Names` (507), `TStrings.Values` (512)

4.74.65 TStrings.Options

Synopsis: A set of `TStringsOption` (298) - various boolean properties.

Declaration: `Property Options : TStringsOptions`

Visibility: public

Access: Read,Write

Description: Set `Options` instead of the underlying boolean properties.

See also: `TStringsOption` (298)

4.74.66 TStrings QuoteChar

Synopsis: Quote character used in DelimitedText (506).

Declaration: `Property QuoteChar : Char`

Visibility: public

Access: Read,Write

Description: `QuoteChar` is the character used by the `DelimitedText` (506) property to quote strings that have a space or non-printing character in it.

4.74.67 TStrings.SkipLastLineBreak

Synopsis: Do not add a linebreak to the last item.

Declaration: `Property SkipLastLineBreak : Boolean`

Visibility: public

Access: Read,Write

Description: `SkipLastLineBreak` can be set to `True` to omit a linebreak character after the last string in the `TStrings.Text` (510). This also means when writing to file, that the file will not have a terminating linebreak character.

Note that `SkipLastLineBreak` has the opposite meaning to `TStrings.TrailingLineBreak` (509).

See also: `TStrings.TrailingLineBreak` (509), `TStrings.LineBreak` (507), `TStrings.Text` (510), `TStrings.Options` (508)

4.74.68 TStrings.TrailingLineBreak

Synopsis: Add a linebreak to the last item.

Declaration: `Property TrailingLineBreak : Boolean`

Visibility: public

Access: Read,Write

Description: `TrailingLineBreak` can be set to `False` to omit a linebreak character after the last string in the `TStrings.Text` (510). This also means when writing to file, that the file will not have a terminating linebreak character.

Note that `TrailingLineBreak` has the opposite meaning to `TStrings.SkipLastLineBreak` (509).

See also: `TStrings.SkipLastLineBreak` (509), `TStrings.LineBreak` (507), `TStrings.Text` (510), `TStrings.Options` (508)

4.74.69 TStrings.StrictDelimiter

Synopsis: Should only the delimiter character be considered a delimiter.

Declaration: `Property StrictDelimiter : Boolean`

Visibility: public

Access: Read,Write

Description: `StrictDelimiter` can be used to indicate that only the delimiter character should be considered a delimiter when setting `DelimitedText` (506): under normal circumstances, quotes and spaces are considered specially (see the `TStrings.CommaText` (504) property for more information).

When `StrictDelimiter` is set to `True` then only the `Delimiter` (506) character is considered when splitting the text in items: no quoting whatsoever is performed when writing the `DelimitedText` property. However, when reading the `DelimitedText` property, quoted strings are taken into account (so a quoted string can contain a delimiter that is treated as text instead of a delimiter).

See also: `DelimitedText` (506), `CommaText` (504), `Delimiter` (506), `TStrings.Options` (508)

4.74.70 TStrings.Strings

Synopsis: Indexed access to the strings in the list.

Declaration: `Property Strings[Index: Integer]: string; default`

Visibility: public

Access: Read,Write

Description: `Strings` is the default property of `TStrings`. It provides indexed read-write access to the list of strings. Reading it will return the string at position `Index` in the list. Writing it will set the string at position `Index`.

`Index` is the position of the string in the list. It is zero-based, i.e. valued values range from 0 (the first string in the list) till `Count-1` (the last string in the list). When browsing through the strings in the list, this fact must be taken into account.

To access the objects associated with the strings in the list, use the `TStrings.Objects` (508) property. The name parts of name-value pairs can be accessed with the `TStrings.Names` (507) property, and the values can be set or read through the `TStrings.Values` (512) property.

Searching through the list can be done using the `TStrings.IndexOf` (497) method.

Errors: If `Index` is outside the allowed range, an `EStringListError` (322) exception is raised.

See also: `TStrings.Count` (505), `TStrings.Objects` (508), `TStrings.Names` (507), `TStrings.Values` (512), `TStrings.IndexOf` (497)

4.74.71 TStrings.StringsAdapter

Synopsis: Not implemented in Free Pascal.

Declaration: `Property StringsAdapter : IStringsAdapter`

Visibility: public

Access: Read,Write

Description: Not implemented in Free Pascal.

4.74.72 TStrings.Text

Synopsis: Contents of the list as one big string.

Declaration: `Property Text : string`

Visibility: public

Access: Read,Write

Description: `Text` returns, when read, the contents of the stringlist as one big string consisting of all strings in the list, separated by an end-of-line marker. When this property is set, the string will be cut into smaller strings, based on the positions of end-of-line markers in the string. Any previous content of the stringlist will be lost.

Remark If any of the strings in the list contains an end-of-line marker, then the resulting string will appear to contain more strings than actually present in the list. To avoid this ambiguity, use the `TStrings.CommaText` (504) property instead.

See also: `TStrings.Strings` (510), `TStrings.Count` (505), `TStrings.CommaText` (504)

4.74.73 TStrings.TextLineBreakStyle

Synopsis: Determines which line breaks to use in the `Text` (510) property.

Declaration: `Property TextLineBreakStyle : TTextLineBreakStyle`

Visibility: public

Access: Read,Write

Description: `TextLineBreakStyle` determines which linebreak style is used when constructing the `Text` property: the same rules are used as in the writing to text files:

tlbsLF Lines are separated with a linefeed character #10.

tlbsCRLF Lines are separated with a carriage-return/linefeed character pair: #13#10.

tlbsCR Lines are separated with a carriage-return character #13.

It has no effect when setting the text property.

See also: `Text` (510)

4.74.74 TStrings.UseLocale

Synopsis: Determines what methods are used in strings comparison.

Declaration: `Property UseLocale : Boolean`

Visibility: public

Access: Read,Write

Description: Set `UseLocale` to `True` to use `AnsiCompareStr` (281) and `AnsiCompareText` (281) for comparing strings in the `TStrings` object.

Set `UseLocale` to `False` to use `CompareStr` (281) and `CompareText` (281) for comparing strings in the `TStrings` object.

The default value is `True`.

See also: `TStrings.Options` (508)

4.74.75 TStrings.ValueFromIndex

Synopsis: Return the value part of a string based on it's index.

Declaration: `Property ValueFromIndex[Index: Integer]: string`

Visibility: public

Access: Read,Write

Description: `ValueFromIndex` returns the value part of a string based on the string index. The value part are all characters in the string after the `NameValueSeparator` (508) character, or all characters if the `NameValueSeparator` character is not present.

4.74.76 TStrings.Values

Synopsis: Value parts of the name-value pairs in the list.

Declaration: `Property Values[Name: string]: string`

Visibility: public

Access: Read,Write

Description: `Values` represents the value parts of the name-value pairs in the list.

When reading this property, if there is a name-value pair in the list of strings that has name part `Name`, then the corresponding value is returned. If there is no such pair, an empty string is returned.

When writing this value, first it is checked whether there exists a name-value pair in the list with name `Name`. If such a pair is found, it's value part is overwritten with the specified value. If no such pair is found, a new name-value pair is added with the specified `Name` and value.

Remark

- 1.Names are compared case-insensitively.
- 2.Any character, including whitespace, up till the first equal (=) sign in a string is considered part of the name.

See also: `TStrings.Names` (507), `TStrings.Strings` (510), `TStrings.Objects` (508)

4.74.77 TStrings.WriteBOM

Synopsis: Write BOM when writing stringlist to stream.

Declaration: `Property WriteBOM : Boolean`

Visibility: public

Access: Read,Write

Description: `WriteBOM` signals the stringlist to write a BOM Marker to the stream if the stringlist is written to stream using `TStrings.SaveToStream` (502) or `TStrings.SaveToFile` (502)

See also: `TStrings.SaveToStream` (502), `TStrings.SaveToFile` (502), `TStrings.Options` (508)

4.75 TStringsEnumerator

4.75.1 Description

`TStringsEnumerator` implements the `#rtl.system.IEnumerator` (1591) interface for the `TStrings` (486) class, so the `TStrings` class can be used in a `for ... in` loop. It is returned by the `TStrings.GetEnumerator` (497) method of `TStrings`.

See also: `TStrings` (486), `TStrings.GetEnumerator` (497), `#rtl.system.IEnumerator` (1591)

4.75.2 Method overview

Page	Method	Description
513	<code>Create</code>	Initialize a new instance of <code>TStringsEnumerator</code> .
513	<code>GetCurrent</code>	Return the current pointer in the list.
513	<code>MoveNext</code>	Move the position of the enumerator to the next position in the list.

4.75.3 Property overview

Page	Properties	Access	Description
514	<code>Current</code>	<code>r</code>	Current pointer in the list.

4.75.4 TStringsEnumerator.Create

Synopsis: Initialize a new instance of `TStringsEnumerator`.

Declaration: `constructor Create(AStrings: TStrings)`

Visibility: `public`

Description: `Create` initializes a new instance of `TStringsEnumerator` and keeps a reference to the stringlist `AStrings` that will be enumerated.

See also: `TStrings` (486)

4.75.5 TStringsEnumerator.GetCurrent

Synopsis: Return the current pointer in the list.

Declaration: `function GetCurrent : string`

Visibility: `public`

Description: `GetCurrent` returns the current string item in the enumerator.

Errors: No checking is done on the validity of the current position.

See also: `MoveNext` (513), `TStringItem` (318)

4.75.6 TStringsEnumerator.MoveNext

Synopsis: Move the position of the enumerator to the next position in the list.

Declaration: `function MoveNext : Boolean`

Visibility: public

Description: `MoveNext` puts the pointer on the next item in the stringlist, and returns `True` if this succeeded, or `False` if the pointer is past the last element in the list.

Errors: Note that if `False` is returned, calling `GetCurrent` will result in an exception.

See also: `GetCurrent` ([513](#))

4.75.7 TStringsEnumerator.Current

Synopsis: Current pointer in the list.

Declaration: `Property Current : string`

Visibility: public

Access: Read

Description: `Current` redefines `GetCurrent` ([513](#)) as a property.

See also: `GetCurrent` ([513](#))

4.76 TStringStream

4.76.1 Description

`TStringStream` stores its data in an `ansistring`. The contents of this string is available as the `DataStream` ([517](#)) property. It also introduces some methods to read or write parts of the stringstream's data as a string.

The main purpose of a `TStringStream` is to be able to treat a string as a stream from which can be read.

See also: `TStream` ([466](#)), `TStringStream.DataStream` ([517](#)), `TStringStream.ReadString` ([517](#)), `TStringStream.WriteString` ([517](#))

4.76.2 Method overview

Page	Method	Description
515	<code>Create</code>	Creates a new stringstream and sets its initial content.
515	<code>CreateRaw</code>	Create stringstream using codepage of string.
515	<code>Destroy</code>	Free the instance of the stream.
516	<code>ReadAnsiString</code>	Read an <code>ansistring</code> from the stream.
517	<code>ReadString</code>	Reads a string of length <code>Count</code> .
516	<code>ReadUnicodeString</code>	Read a unicode string from the stream.
516	<code>WriteAnsiString</code>	Write an <code>ansistring</code> to the stream.
517	<code>WriteString</code>	<code>WriteString</code> writes a string to the stream.
516	<code>WriteUnicodeString</code>	Write a unicode string to the stream.

4.76.3 Property overview

Page	Properties	Access	Description
517	DataStream	r	Contains the contents of the stream in string form.
518	Encoding	r	Encoding of the string with the data.
518	OwnsEncoding	r	Does the stream instance owns the encoding.
517	UnicodeDataStream	r	Datastring as unicode string.

4.76.4 TStringStream.Create

Synopsis: Creates a new stringstream and sets its initial content.

Declaration: `constructor Create(const ABytes: TBytes); Override; Overload`
`constructor Create(const AString: string); Overload`
`constructor Create(const AString: string; AEncoding: TEncoding;`
`AOwnsEncoding: Boolean); Overload`
`constructor Create(const AString: string; ACodePage: Integer); Overload`
`constructor Create(const AString: UnicodeString); Overload`
`constructor Create(const AString: UnicodeString; AEncoding: TEncoding;`
`AOwnsEncoding: Boolean); Overload`
`constructor Create(const AString: UnicodeString; ACodePage: Integer)`
`; Overload`

Visibility: public

Description: `Create` creates a new `TStringStream` instance and sets its initial content to `AString`. The position is still 0 but the size of the stream will equal the length of the string.

The `Encoding` argument specifies the codepage with which the bytes in the string will be interpreted. If it is not specified (or `Nil`) the system default encoding will be used. The correct encoding can be detected from the string by using the `TStringStream.CreateRaw` ([515](#)) constructor.

See also: `TStringStream.DataString` ([517](#)), `TStringStream.UnicodeDataStream` ([517](#)), `TStringStream.Encoding` ([518](#)), `TStringStream.CreateRaw` ([515](#))

4.76.5 TStringStream.CreateRaw

Synopsis: Create stringstream using codepage of string.

Declaration: `constructor CreateRaw(const AString: RawByteString); Overload`

Visibility: public

Description: `CreateRaw` will create the stream using the codepage of the string passed on in the constructor. The default constructor uses the default codepage of the system.

See also: `TStringStream.Create` ([515](#))

4.76.6 TStringStream.Destroy

Synopsis: Free the instance of the stream.

Declaration: `destructor Destroy; Override`

Visibility: public

Description: `Destroy` frees the encoding if it owns it and calls the inherited destructor.

4.76.7 TStringStream.ReadUnicodeString

Synopsis: Read a unicode string from the stream.

Declaration: `function ReadUnicodeString(Count: LongInt) : UnicodeString`

Visibility: public

Description: `ReadUnicodeString` will read a unicodestring and correctly handle the codepage translation if necessary..

See also: `TStream.ReadString` (466), `TStream.ReadAnsiString` (474), `TStringStream.WriteUnicodeString` (516)

4.76.8 TStringStream.WriteUnicodeString

Synopsis: Write a unicode string to the stream.

Declaration: `procedure WriteUnicodeString(const AString: UnicodeString)`

Visibility: public

Description: `WriteUnicodeString` will write a unicodestring and correctly handle the codepage translation if necessary.

See also: `TStream.ReadString` (466), `TStringStream.ReadUnicodeString` (516), `TStringStream.ReadAnsiString` (516), `TStringStream.WriteAnsiString` (516)

4.76.9 TStringStream.ReadAnsiString

Synopsis: Read an ansistring from the stream.

Declaration: `function ReadAnsiString(Count: LongInt) : AnsiString; Overload`

Visibility: public

Description: `ReadAnsiString` will read an ansistring and correctly handle the codepage translation if necessary..

See also: `TStream.ReadString` (466), `TStringStream.WriteUnicodeString` (516), `TStringStream.WriteAnsiString` (516), `TStringStream.ReadUnicodeString` (516)

4.76.10 TStringStream.WriteAnsiString

Synopsis: Write an ansistring to the stream.

Declaration: `procedure WriteAnsiString(const AString: AnsiString); Override`

Visibility: public

Description: `WriteAnsiString` will write an ansistring and correctly handle the codepage translation if necessary.

See also: `TStream.ReadString` (466), `TStringStream.ReadUnicodeString` (516), `TStringStream.ReadAnsiString` (516), `TStringStream.WriteUnicodeString` (516)

4.76.11 TStringStream.ReadString

Synopsis: Reads a string of length Count.

Declaration: `function ReadString(Count: LongInt) : string`

Visibility: public

Description: `ReadString` reads Count bytes from the stream and returns the read bytes as a string. If less than Count bytes were available, the string has as many characters as bytes could be read.

The `ReadString` method is a wrapper around the `Read` (514) method. It does not do the same string as the `TStream.ReadAnsiString` (474) method, which first reads a length integer to determine the length of the string to be read.

See also: `TStringStream.Read` (514), `TStream.ReadAnsiString` (474)

4.76.12 TStringStream.WriteString

Synopsis: `WriteString` writes a string to the stream.

Declaration: `procedure WriteString(const AString: string)`

Visibility: public

Description: `WriteString` writes a string to the stream.

4.76.13 TStringStream.DataString

Synopsis: Contains the contents of the stream in string form.

Declaration: `Property DataString : string`

Visibility: public

Access: Read

Description: Contains the contents of the stream in string form.

4.76.14 TStringStream.UnicodeDataString

Synopsis: Datastring as unicode string.

Declaration: `Property UnicodeDataString : UnicodeString`

Visibility: public

Access: Read

Description: `UnicodeDataString` returns the data string as a unicode encoded string.

See also: `TStringStream.DataString` (517)

4.76.15 TStringStream.OwnsEncoding

Synopsis: Does the stream instance owns the encoding.

Declaration: `Property OwnsEncoding : Boolean`

Visibility: `public`

Access: `Read`

Description: `OwnsEncoding` indicates whether the stream instance owns the encoding or not. If it owns the encoding, it will be freed when the stream instance is freed.

See also: `TStringStream.Encoding` ([518](#))

4.76.16 TStringStream.Encoding

Synopsis: Encoding of the string with the data.

Declaration: `Property Encoding : TEncoding`

Visibility: `public`

Access: `Read`

Description: `Encoding` is the encoding of the string which contains the data. If the encoding was not specified (or detected) in the constructor, this is the default system encoding.

See also: `TStringStream.OwnsEncoding` ([518](#)), `TStringStream.Create` ([515](#)), `TStringStream.CreateRaw` ([515](#))

4.77 TTextObjectWriter

4.77.1 Description

Not yet implemented.

4.78 TThread

4.78.1 Description

The `TThread` class encapsulates the native thread support of the operating system. To create a thread, declare a descendant of the `TThread` object and override the `Execute` ([520](#)) method. In this method, the `tthread`'s code should be executed. To run a thread, create an instance of the `tthread` descendant, and call its `execute` method.

It is also possible to simply execute a method or static procedure in a thread using the `TThread.ExecuteInThread` ([524](#)) class method.

See also: `EThread` ([322](#)), `TThread.Execute` ([520](#)), `TThread.ExecuteInThread` ([524](#))

4.78.2 Method overview

Page	Method	Description
529	AfterConstruction	Code to be executed after construction but before execute.
522	CheckTerminated	Check if the current thread has finished executing.
521	Create	Creates a new thread.
521	CreateAnonymousThread	Execute code in an anonymous thread.
530	CurrentIsMain	
521	Destroy	Destroys the thread object.
520	Execute	Execute method. Must be overridden in a descendant thread.
524	ExecuteInThread	Execute a method or static procedure in a thread.
520	ForceQueue	Always queue a method for execution in the main thread.
524	GetCPUUsage	
523	GetSystemTimes	Return CPU stats.
524	GetTickCount	Return tick count (32-bit).
524	GetTickCount64	Return tick count (64-bit).
521	NameThreadForDebugging	Set a thread name.
520	Queue	Queue a method for execution in the main thread.
522	RemoveQueuedEvents	Remove methods scheduled for execution from queue.
529	Resume	Resumes the thread's execution. Deprecated, see <code>TThread.Start</code> .
522	SetReturnValue	Set return value of a thread.
523	Sleep	Prevent thread execution.
523	SpinWait	Prevent thread execution in a spin-wait loop.
529	Start	Starts a thread that was created in a suspended state.
530	Suspend	Suspends the thread's execution.
520	Synchronize	Synchronizes the thread by executing the method in the main thread.
530	Terminate	Signals the thread it should terminate.
530	WaitFor	Waits for the thread to terminate and returns the exit status.
523	Yield	Yield execution to other threads.

4.78.3 Property overview

Page	Properties	Access	Description
531	Current	r	
530	CurrentThread	r	Return current thread instance.
532	ExternalThread	r	Is the thread instance an external thread ?
534	FatalException	r	Exception that occurred during thread execution.
533	Finished	r	Has the thread finished executing.
532	FreeOnTerminate	rw	Indicates whether the thread should free itself when it stops executing.
532	Handle	r	Returns the thread handle.
531	IsSingleProcessor	r	Is the current system single processor or not.
534	OnTerminate	rw	Event called when the thread terminates.
533	Priority	rw	Returns the thread priority.
531	ProcessorCount	r	Return the processor count for this system.
533	Suspended	rw	Indicates whether the thread is suspended.
533	ThreadID	r	Returns the thread ID.

4.78.4 TThread.Execute

Synopsis: Execute method. Must be overridden in a descendant thread.

Declaration: `procedure Execute; Virtual; Abstract`

Visibility: protected

Description: `Execute` is a method that must be overridden in descendant classes of the thread. It must contain the code that must execute in the thread. The `Execute` method is responsible for checking `Terminated` (281) at regular intervals: when it is set to `True` the execute method must exit.

See also: `Terminated` (281)

4.78.5 TThread.Synchronize

Synopsis: Synchronizes the thread by executing the method in the main thread.

Declaration: `procedure Synchronize (AMethod: TThreadMethod)`
`class procedure Synchronize (AThread: TThread; AMethod: TThreadMethod)`

Visibility: protected

Description: Synchronizes the thread by executing the method in the main thread.

4.78.6 TThread.Queue

Synopsis: Queue a method for execution in the main thread.

Declaration: `procedure Queue (aMethod: TThreadMethod)`
`class procedure Queue (AThread: TThread; aMethod: TThreadMethod); Static`

Visibility: protected

Description: `Queue` schedules a method `aMethod` for execution in the main thread. In difference with `TThread.Synchronize` (520), `Queue` just posts the method for execution in a queue, and does not wait for it to be executed, so this call returns at once.

If the call is made in the main thread, the method is not put on the queue, but is executed at once.

In the class procedure overloaded version of this call, the thread for which the method must be posted is the first argument. In the protected version of this call (used in the `tthread` instance), this argument is not there, and the thread instance is used.

When a thread object is destroyed (after it has finished executing) all its queued calls are removed from the queue list.

See also: `TThread.Synchronize` (520), `TThread.RemoveQueuedEvents` (522), `TThread.ForceQueue` (520)

4.78.7 TThread.ForceQueue

Synopsis: Always queue a method for execution in the main thread.

Declaration: `procedure ForceQueue (aMethod: TThreadMethod)`
`class procedure ForceQueue (AThread: TThread; aMethod: TThreadMethod)`
`; Static`

Visibility: protected

Description: `ForceQueue` schedules a method `aMethod` for execution in the main thread, just like `TThread.Queue` (520). In difference with `TThread.Queue` (520), if the call is made from the main thread, the function will not be executed at once, but will actually be put in the queue.

See also: `TThread.Synchronize` (520), `TThread.RemoveQueuedEvents` (522), `TThread.Queue` (520)

4.78.8 TThread.Create

Synopsis: Creates a new thread.

Declaration: `constructor Create(CreateSuspended: Boolean; const StackSize: SizeUInt)`

Visibility: public

Description: Creates a new thread.

4.78.9 TThread.Destroy

Synopsis: Destroys the thread object.

Declaration: `destructor Destroy; Override`

Visibility: public

Description: Destroys the thread object.

4.78.10 TThread.CreateAnonymousThread

Synopsis: Execute code in an anonymous thread.

Declaration: `class function CreateAnonymousThread(aProc: TProcedure) : TThread
; Static`

Visibility: public

Description: `CreateAnonymousThread` will create an instance of a `TThread` descendant and calls `aProc` in this procedure. This can be used to quickly execute a method in another thread without having to explicitly declare a thread for such purposes. It returns the created `TThread` instance, which can be checked for termination etc.

Note that this method differs slightly from Delphi in that FPC does not yet support anonymous methods, so the signature of `aProc` differs slightly.

See also: `TThread.CheckTerminated` (522)

4.78.11 TThread.NameThreadForDebugging

Synopsis: Set a thread name.

Declaration: `class procedure NameThreadForDebugging(aThreadName: UnicodeString;
aThreadID: TThreadID); Static
class procedure NameThreadForDebugging(aThreadName: AnsiString;
aThreadID: TThreadID); Static`

Visibility: public

Description: `NameThreadForDebugging` sets the name of thread `aThreadID` to `aThreadName`. The thread name can be Unicode or an string. This is mainly useful for debugging purposes, as thread names are more easily recognizable than IDs.

Note that this requires OS support, so at the moment it is Windows and Unix only.

4.78.12 `TThread.SetReturnValue`

Synopsis: Set return value of a thread.

Declaration: `class procedure SetReturnValue(aValue: Integer); Static`

Visibility: public

Description: `TThread.SetReturnValue` sets the return value of an internally created thread.

Errors: If the thread was not created by the FPC program, an `EThreadExternalException` (322) exception is raised.

See also: `EThreadExternalException` (322), `TThread.CheckTerminated` (522)

4.78.13 `TThread.CheckTerminated`

Synopsis: Check if the current thread has finished executing.

Declaration: `class function CheckTerminated : Boolean; Static`

Visibility: public

Description: `TThread.CheckTerminated` can be used to check if the current thread has finished executing (i.e. `Execute` has finished). This can be called from methods in other classes where the current thread instance is not available.

Errors: If the thread was not created by the FPC program, an `EThreadExternalException` (322) exception is raised.

See also: `EThreadExternalException` (322), `TThread.SetReturnValue` (522)

4.78.14 `TThread.RemoveQueuedEvents`

Synopsis: Remove methods scheduled for execution from queue.

Declaration: `class procedure RemoveQueuedEvents(aThread: TThread;
aMethod: TThreadMethod); Static
class procedure RemoveQueuedEvents(aMethod: TThreadMethod); Static
class procedure RemoveQueuedEvents(aThread: TThread); Static`

Visibility: public

Description: `RemoveQueuedEvents` removes methods from the list of methods waiting for execution in the main thread.

If only `aThread` is specified, all methods scheduled for execution by that thread are removed.

If only `aMethod` is specified, then all calls to that method are removed, regardless of the thread. The methods are compared with both the Code and Data pointers (method code and method object).

If both arguments are specified, then all calls to the given method by the given thread are removed. This is a known difference to Delphi that deletes all given methods or all methods by the given thread.

See also: `TThread.Synchronize` (520), `TThread.Queue` (520)

4.78.15 TThread.SpinWait

Synopsis: Prevent thread execution in a spin-wait loop.

Declaration: `class procedure SpinWait(aIterations: LongWord); Static`

Visibility: public

Description: `SpinWait` blocks the execution of the thread in a spin-wait loop: it simply executes some simple instructions.

This can be used to create short time delays without an immediate thread switch (e.g. `SysUtils.Sleep` (281) can cause a thread switch). The input parameter (alterations) specifies the number of spin loops.

See also: `SysUtils.Sleep` (281), `TThread.Sleep` (523)

4.78.16 TThread.Sleep

Synopsis: Prevent thread execution.

Declaration: `class procedure Sleep(aMilliseconds: Cardinal); Static`

Visibility: public

Description: `Sleep` blocks the execution of the thread for `aMilliseconds`. This function simply calls `sysutils.sleep` (281)

In difference with `TThread.SpinWait` (523), a thread switch may occur during the sleep.

See also: `SysUtils.Sleep` (281), `TThread.SpinWait` (523)

4.78.17 TThread.Yield

Synopsis: Yield execution to other threads.

Declaration: `class procedure Yield; Static`

Visibility: public

Description: `TThread.Yield` yields the processor to other threads. It can be called from methods outside the thread class itself.

4.78.18 TThread.GetSystemTimes

Synopsis: Return CPU stats.

Declaration: `class function GetSystemTimes(out aSystemTimes: TSystemTimes) : Boolean
; Static`

Visibility: public

Description: `GetSystemTimes` is provided for Delphi compatibility only, it currently returns empty values only.

See also: `TSystemTimes` (281)

4.78.19 TThread.GetCPUUsage

Declaration: `class function GetCPUUsage(var Previous: TSystemTimes) : Integer`

Visibility: public

4.78.20 TThread.GetTickCount

Synopsis: Return tick count (32-bit).

Declaration: `class function GetTickCount : LongWord; Static`

Visibility: public

Description: `GetTickCount` is deprecated and should not be used. Use `TThread.GetTickCount64` ([524](#)) instead.

See also: `TThread.GetTickCount64` ([524](#))

4.78.21 TThread.GetTickCount64

Synopsis: Return tick count (64-bit).

Declaration: `class function GetTickCount64 : QWord; Static`

Visibility: public

Description: `GetTickCount64` simply calls `SysUtils.GetTickCount64` ([281](#)) and is implemented for Delphi compatibility only.

See also: `SysUtils.GetTickCount64` ([281](#))

4.78.22 TThread.ExecuteInThread

Synopsis: Execute a method or static procedure in a thread.

```
Declaration: class function ExecuteInThread(AMethod: TThreadExecuteHandler;
                                           AOnTerminate: TNotifyEvent) : TThread
                                           ; Overload; Static
class function ExecuteInThread(AMethod: TThreadExecuteStatusHandler;
                               AOnStatus: TThreadStatusNotifyEvent;
                               AOnTerminate: TNotifyEvent) : TThread
                               ; Overload; Static
class function ExecuteInThread(AMethod: TThreadExecuteCallBack;
                               AData: Pointer;
                               AOnTerminate: TNotifyCallBack) : TThread
                               ; Overload; Static
class function ExecuteInThread(AMethod: TThreadExecuteStatusCallBack;
                               AOnStatus: TThreadStatusNotifyCallBack;
                               AData: Pointer;
                               AOnTerminate: TNotifyCallBack) : TThread
                               ; Overload; Static
```

Visibility: public

Description: `TThread.ExecuteInThread` is a class method which allows to quickly execute a method or procedure in a thread. The method or procedure to be executed is passed in `Method`, this can be a method or a plain (static) procedure.

The caller can be notified of thread termination: In the optional argument `AOnTerminate` a callback (procedure or method, depending on the signature) can be specified that will be called when the thread terminated. This callback is executed in the main thread.

The signature of `AMethod` determines whether status reporting is enabled or not. If the method of type `TThreadExecuteStatusHandler` (299) or `TThreadExecuteStatusCallback` (299), then an extra `AOnStatus` callback must be specified. This callback will be called in the main thread whenever the thread wishes to be report its status. The status callback should not do extensive work, because while the status callback is called, thread execution is suspended.

When using a plain procedure, extra data can be passed on to the procedure in `AData`. The `AData` pointer will be passed to the thread method, and also to the thread status callback and thread termination callback.

See also: `TThreadExecuteHandler` (299), `TThreadExecuteStatusHandler` (299), `TThreadExecuteStatusCallback` (299), `TThreadExecuteCallback` (299)

Listing: `./examples/classesex/tthrc.pp`

```

program tthrc;

uses cthreads, sysutils, classes;

Var
  D : Integer;

Procedure DoneThread(Sender : TObject; AData : Pointer);

begin
  Writeln('Thread ', TThread(Sender).ThreadID, ' done. D is currently: ', PInteger(AData)^);
end;

Procedure DoThread(AData : Pointer);

Var
  I : integer;

begin
  for I:=1 to 10 do
    begin
      Sleep(10*Random(30));
      Writeln('Thread ', TThread.CurrentThread.ThreadID, ' ping ', I);
      Inc(PInteger(AData)^, i);
    end;
  end;

Var
  T1, T2 : TThread;

begin
  T1:=TThread.ExecuteInThread(@DoThread, @D, @DoneThread);
  T2:=TThread.ExecuteInThread(@DoThread, @D, @DoneThread);
  Writeln('Main thread done');
  T1.WaitFor;
  T2.WaitFor;
end.

```

Listing: ./examples/classesex/tthre.pp

```

program tthre;

{$mode objfpc}
{$H+}

uses cthreads, sysutils, classes;

Type
  TTestThread = Class(TObject)
    D : Integer;
    Procedure DoneThread(Sender : TObject);
    Procedure DoThread;
    Procedure Run;
  end;

Procedure TTestThread.DoneThread(Sender : TObject);

begin
  Writeln('Thread ',TThread(Sender).ThreadID,' done. D is currently: ', D);
end;

Procedure TTestThread.DoThread;

Var
  I : integer;

begin
  for I:=1 to 10 do
    begin
      Sleep(10*Random(30));
      Writeln('Thread ',TThread.CurrentThread.ThreadID,' ping ',I);
      Inc(D,I);
    end;
  end;

Procedure TTestThread.Run;

Var
  T1,T2 : TThread;

begin
  T1:=TThread.ExecuteInThread(@DoThread,@DoneThread);
  T2:=TThread.ExecuteInThread(@DoThread,@DoneThread);
  Writeln('Main thread done');
  T1.WaitFor;
  T2.WaitFor;
end;

begin
  With TTestThread.Create do
    try
      Run;
    finally

```

```

    Free;
  end;
end.

```

Listing: ./examples/classesex/tthrcs.pp

```

program tthrcs;
{$h+}
uses cthreads, sysutils, classes;

Var
  D : Integer;
  DoneThreads : Integer;

Procedure DoneThread(Sender : TObject; Data : Pointer);

begin
  Inc(DoneThreads);
  Writeln('Thread ', TThread(Sender).ThreadID, ' done. D is currently: ', PInteger(Data)^);
end;

Procedure ReportThreadStatus(Sender : TThread; AData : Pointer; Const status : String);

begin
  Writeln('Thread ', Sender.ThreadID, ' Status report : ', Status);
end;

Procedure DoThread(AData : Pointer; Report : TThreadReportStatus);

Var
  I : integer;

begin
  for I:=1 to 10 do
    begin
      Sleep(10*Random(30));
      Report('Ping '+IntToStr(i));
      Inc(PInteger(AData)^, i);
    end;
  end;

Var
  T1, T2 : TThread;

begin
  DoneThreads:=0;
  T1:=TThread.ExecuteInThread(@DoThread, @ReportThreadStatus, @D, @DoneThread);
  T2:=TThread.ExecuteInThread(@DoThread, @ReportThreadStatus, @D, @DoneThread);
  Writeln('Main thread loop');
  While DoneThreads<2 do
    begin
      Sleep(10);
      CheckSynchronize;
    end;
  T1.WaitFor;
  T2.WaitFor;
end.

```

Listing: ./examples/classesex/tthres.pp

```

program tthrc;

{$mode objfpc}
{$H+}

uses cthreads, sysutils, classes;

Type
  TTestThread = Class(TObject)
    D : Integer;
    DoneThreads : integer;
    Procedure DoneThread(Sender : TObject);
    Procedure ReportThreadStatus(Sender : TThread; Const status : String);
    Procedure DoThread(Report: TThreadReportStatus);
    Procedure Run;
  end;

Procedure TTestThread.DoneThread(Sender : TObject);

begin
  Inc(DoneThreads);
  WriteLn('Thread ',TThread(Sender).ThreadID,' done. D is currently: ', D);
end;

Procedure TTestThread.ReportThreadStatus(Sender : TThread; Const status : String);

begin
  WriteLn('Thread ',Sender.ThreadID,' Status report : ',Status);
end;

Procedure TTestThread.DoThread(Report : TThreadReportStatus);

Var
  I : integer;

begin
  for I:=1 to 10 do
    begin
      Sleep(10*Random(30));
      Report('Ping ' + IntToStr(i));
      Inc(D,i);
    end;
  end;

Procedure TTestThread.Run;

Var
  T1,T2 : TThread;

begin
  DoneThreads:=0;
  T1:=TThread.ExecuteInThread(@DoThread,@ReportThreadStatus,@DoneThread);
  T2:=TThread.ExecuteInThread(@DoThread,@ReportThreadStatus,@DoneThread);
  WriteLn('Main thread loop');
  While DoneThreads<2 do

```

```

    begin
    Sleep(10);
    CheckSynchronize;
    end;
    T1.WaitFor;
    T2.WaitFor;
end;

begin
  With TTestThread.Create do
    try
      Run;
    finally
      Free;
    end;
  end;
end.

```

4.78.23 TThread.AfterConstruction

Synopsis: Code to be executed after construction but before execute.

Declaration: `procedure AfterConstruction; Override`

Visibility: `public`

Description: `AfterConstruction` is overridden in `TThread`, it actually starts the thread if it was created with `CreateSuspended` equal to `False`, i.e. not suspended. When overriding this method, the inherited method must be called, or the thread will never be started.

4.78.24 TThread.Start

Synopsis: Starts a thread that was created in a suspended state.

Declaration: `procedure Start`

Visibility: `public`

Description: The effect of this method is currently the same as calling `TThread.Resume` after creating a thread in a suspended state. This method was added for Delphi-compatibility, where it was introduced after `TThread.Suspend` and `TThread.Resume` were deprecated.

See also: `TThread.Create` ([521](#))

4.78.25 TThread.Resume

Synopsis: Resumes the thread's execution. Deprecated, see `TThread.Start`.

Declaration: `procedure Resume`

Visibility: `public`

Description: Resumes the thread's execution. Deprecated, see `TThread.Start`.

See also: `TThread.Start` ([529](#)), `TThread.Suspend` ([530](#))

4.78.26 TThread.Suspend

Synopsis: Suspends the thread's execution.

Declaration: `procedure Suspend`

Visibility: `public`

Description: On non-Windows platforms, a thread can only suspend itself. Other threads can wake up a suspended thread by calling `TThread.Start`.

See also: `TThread.Resume` ([529](#)), `TThread.Start` ([529](#))

4.78.27 TThread.Terminate

Synopsis: Signals the thread it should terminate.

Declaration: `procedure Terminate`

Visibility: `public`

Description: `Terminate` sets the `TThread.Terminated` ([518](#)) flag. It does not in any way attempt to terminate the thread in any other way, this just signals the thread that it should stop executing at the earliest possible moment.

See also: `TThread.Terminated` ([518](#)), `TThread.WaitFor` ([530](#)), `TThread.FreeOnTerminate` ([532](#)), `OnTerminate` ([534](#))

4.78.28 TThread.WaitFor

Synopsis: Waits for the thread to terminate and returns the exit status.

Declaration: `function WaitFor : Integer`

Visibility: `public`

Description: `WaitFor` waits for the thread to terminate, and returns the exit status. Note that when executed in the main thread, this method calls `CheckSynchronize` ([304](#)), this is done to avoid deadlocks: if the thread is waiting for a synchronize ([520](#)), then the synchronize methods will be executed and then the `WaitFor` will return.

See also: `TThread.Terminated` ([518](#)), `TThread.WaitFor` ([530](#)), `TThread.FreeOnTerminate` ([532](#)), `TThread.Synchronize` ([520](#))

4.78.29 TThread.CurrentIsMain

Declaration: `class function CurrentIsMain : Boolean; Static`

Visibility: `public`

4.78.30 TThread.CurrentThread

Synopsis: Return current thread instance.

Declaration: `Property CurrentThread : TThread`

Visibility: `public`

Access: Read

Description: `TThread.CurrentThread` can be used to get the current thread instance. This is useful in code that is not inside a `TThread` implementation, but which needs access to the current thread.

For threads that were created outside of FPC code (DLLs or a calling program) this will return a dummy `TThread` instance.

See also: `TThread.ExternalThread` ([532](#))

4.78.31 `TThread.Current`

Declaration: `Property Current : TThread`

Visibility: public

Access: Read

4.78.32 `TThread.ProcessorCount`

Synopsis: Return the processor count for this system.

Declaration: `Property ProcessorCount : LongWord`

Visibility: public

Access: Read

Description: `ProcessorCount` returns the processor count for this system.

Whether this is the number of cores or the number of CPUs present in the hardware, is deliberately unspecified. The number of cores can also vary during the lifetime of the program, and the FPC implementation does not guarantee that this will always match, the value is set at program start.

As such, the number specified should only be used as an indication of how many threads can be executed at once by the system.

See also: `TThread.IsSingleProcessor` ([531](#))

4.78.33 `TThread.IsSingleProcessor`

Synopsis: Is the current system single processor or not.

Declaration: `Property IsSingleProcessor : Boolean`

Visibility: public

Access: Read

Description: `Thread.IsSingleProcessor` returns `True` if `TThread.ProcessorCount` ([531](#)) is less than or equal to 1, `False` otherwise.

See also: `TThread.ProcessorCount` ([531](#))

4.78.34 TThread.FreeOnTerminate

Synopsis: Indicates whether the thread should free itself when it stops executing.

Declaration: `Property FreeOnTerminate : Boolean`

Visibility: `public`

Access: `Read, Write`

Description: `FreeOnTerminate`, when set to `True` indicates that the thread instance will freed automatically as soon as the thread stops executing. You can use the `OnTerminate` (534) property to get a notification of when the thread has terminated and will be freed.

When setting this property to `True`, in general you may not read or write any property of the `TThread` instance from a different thread, because there is no guarantee that the thread instance still exists in memory. This implies 2 things:

1. The `OnTerminate` event handler should be set before setting `FreeOnTerminate` to `True`
2. The properties can still be read and set in the `OnTerminate` event handler, as the thread instance is then still guaranteed to exist.

If `FreeOnTerminate` is set to `False`, to stop and delete a running thread from another thread, the following sample code can be used:

```
aThread.Terminate;
aThread.WaitFor;
FreeAndNil(aThread);
```

See also: `OnTerminate` (534)

4.78.35 TThread.Handle

Synopsis: Returns the thread handle.

Declaration: `Property Handle : TThreadID`

Visibility: `public`

Access: `Read`

Description: Returns the thread handle.

4.78.36 TThread.ExternalThread

Synopsis: Is the thread instance an external thread ?

Declaration: `Property ExternalThread : Boolean`

Visibility: `public`

Access: `Read`

Description: `ExternalThread` returns `True` if the thread is an externally created thread. If the thread was created by the FPC program, this returns `False`. This is useful for examining instances returned by `TThread.CurrentThread` (530).

See also: `TThread.CurrentThread` (530)

4.78.37 TThread.Priority

Synopsis: Returns the thread priority.

Declaration: `Property Priority : TThreadPriority`

Visibility: `public`

Access: `Read, Write`

Description: Returns the thread priority.

4.78.38 TThread.Suspended

Synopsis: Indicates whether the thread is suspended.

Declaration: `Property Suspended : Boolean`

Visibility: `public`

Access: `Read, Write`

Description: Indicates whether the thread is suspended.

4.78.39 TThread.Finished

Synopsis: Has the thread finished executing.

Declaration: `Property Finished : Boolean`

Visibility: `public`

Access: `Read`

Description: `Finished` is `True` when `TThread.Execute` ([520](#)) has finished executing, but the thread is still cleaning up (calling `OnTerminate`, etc).

See also: `TThread.Execute` ([520](#)), `TThread.OnTerminate` ([534](#))

4.78.40 TThread.ThreadID

Synopsis: Returns the thread ID.

Declaration: `Property ThreadID : TThreadID`

Visibility: `public`

Access: `Read`

Description: Returns the thread ID.

4.78.41 TThread.OnTerminate

Synopsis: Event called when the thread terminates.

Declaration: `Property OnTerminate : TNotifyEvent`

Visibility: `public`

Access: `Read, Write`

Description: `TThread.OnTerminate` is called when the thread terminates. The event is always called in the context of the main thread, i.e. using `TThread.Synchronize` (520)

See also: `TThread.Synchronize` (520), `TThread.FreeOnTerminate` (532)

4.78.42 TThread.FatalException

Synopsis: Exception that occurred during thread execution.

Declaration: `Property FatalException : TObject`

Visibility: `public`

Access: `Read`

Description: `FatalException` contains the exception that occurred during the thread's execution.

4.79 TThreadList

4.79.1 Description

`TThreadList` is a thread-safe `TList` (426) implementation. Unlike `TList`, it can be accessed read-write by multiple threads: the list implementation will take care of locking the list when adding or removing items from the list.

See also: `TList` (426)

4.79.2 Method overview

Page	Method	Description
535	<code>Add</code>	Adds an element to the list.
535	<code>Clear</code>	Removes all elements from the list.
535	<code>Create</code>	Creates a new thread-safe list.
535	<code>Destroy</code>	Destroys the list instance.
536	<code>LockList</code>	Locks the list for exclusive access.
536	<code>Remove</code>	Removes an item from the list.
536	<code>UnlockList</code>	Unlocks the list after it was locked.

4.79.3 Property overview

Page	Properties	Access	Description
536	<code>Duplicates</code>	<code>rw</code>	Describes what to do with duplicates.

4.79.4 TThreadList.Create

Synopsis: Creates a new thread-safe list.

Declaration: `constructor Create`

Visibility: `public`

Description: `Create` instantiates a new `TThreadList` instance. It initializes a critical section and an internal list object.

See also: `TThreadList.Destroy` ([535](#))

4.79.5 TThreadList.Destroy

Synopsis: Destroys the list instance.

Declaration: `destructor Destroy; Override`

Visibility: `public`

Description: `Destroy` finalizes the critical section, clears the internal list object and calls the inherited destructor.

See also: `TThreadList.Create` ([535](#))

4.79.6 TThreadList.Add

Synopsis: Adds an element to the list.

Declaration: `procedure Add(Item: Pointer)`

Visibility: `public`

Description: `Add` attempts to lock the list and adds the pointer `Item` to the list. After the pointer was added, the list is unlocked again.

See also: `LockList` ([536](#)), `Clear` ([535](#)), `Remove` ([536](#)), `UnlockList` ([536](#))

4.79.7 TThreadList.Clear

Synopsis: Removes all elements from the list.

Declaration: `procedure Clear`

Visibility: `public`

Description: `Clear` attempts to lock the list and then clears the list; all items are removed from the list. After the list is cleared, it is again unlocked.

See also: `LockList` ([536](#)), `Add` ([535](#)), `Remove` ([536](#)), `UnlockList` ([536](#))

4.79.8 TThreadList.LockList

Synopsis: Locks the list for exclusive access.

Declaration: `function LockList : TList`

Visibility: public

Description: `LockList` locks the thread list for exclusive access. The return value for the method is the internal `TList` instance with pointers to the `TThread` instances for the thread list. `Locklist` uses an internal critical section, so all rules for multiple locking of critical sections apply to `locklist/unlocklist` as well.

See also: `Clear` ([535](#)), `Add` ([535](#)), `Remove` ([536](#)), `UnlockList` ([536](#))

4.79.9 TThreadList.Remove

Synopsis: Removes an item from the list.

Declaration: `procedure Remove(Item: Pointer)`

Visibility: public

Description: `Remove` attempts to lock the list and then removes `Item` from the list. After the item is removed, the list is again unlocked.

See also: `LockList` ([536](#)), `Add` ([535](#)), `Clear` ([535](#)), `UnlockList` ([536](#))

4.79.10 TThreadList.UnlockList

Synopsis: Unlocks the list after it was locked.

Declaration: `procedure UnlockList`

Visibility: public

Description: `UnLockList` unlocks the list when it was locked for exclusive access. `UnLocklist` and `LockList` use an internal critical section, so all rules for multiple locking/unlocking of critical sections apply.

See also: `Clear` ([535](#)), `Add` ([535](#)), `Remove` ([536](#)), `lockList` ([536](#))

4.79.11 TThreadList.Duplicates

Synopsis: Describes what to do with duplicates.

Declaration: `Property Duplicates : TDuplicates`

Visibility: public

Access: Read,Write

Description: `Duplicates` describes what the threadlist should do when a duplicate pointer is added to the list. It is identical in behaviour to the `Duplicates` ([484](#)) property of `TStringList` ([481](#)).

See also: `TDuplicates` ([288](#))

4.80 TWriter

4.80.1 Description

The `TWriter` class is a writer class that implements generic component streaming capabilities, independent of the format of the data in the stream. It uses a driver class `TAbstractObjectWriter` (344) to do the actual reading of data. The interface of the `TWriter` class should be identical to the interface in Delphi.

Note that the `TWriter` design is such that it will write a single component to a stream. It will write all children of this component, but it is not designed to write multiple components in succession to one stream.

It should never be necessary to create an instance of this class directly. Instead, the `TStream.WriteComponent` (471) call should be used.

See also: `TFile` (403), `TWriter` (537), `TAbstractObjectReader` (335)

4.80.2 Method overview

Page	Method	Description
538	Create	Creates a new Writer with a stream and bufsize.
539	DefineBinaryProperty	Callback used when defining and streaming custom properties.
538	DefineProperty	Callback used when defining and streaming custom properties.
538	Destroy	Destroys the writer instance.
538	FlushBuffer	Flush the buffer
539	Write	Write raw data to stream.
539	WriteBoolean	Write boolean value to the stream.
540	WriteChar	Write a character to the stream.
539	WriteCollection	Write a collection to the stream.
539	WriteComponent	Stream a component to the stream.
541	WriteCurrency	Write a currency value to the stream.
541	WriteDate	Write a date to the stream.
540	WriteDescendent	Write descendent to stream.
540	WriteFloat	Write a float to the stream.
541	WriteIdent	Write an identifier to the stream.
541	WriteInteger	Write an integer to the stream.
542	WriteListBegin	Write a start-of-list marker to the stream.
542	WriteListEnd	Write an end-of-list marker to the stream.
542	WriteRootComponent	Write a root component to the stream.
541	WriteSet	Write a set value to the stream.
542	WriteSignature	Write a signature to the stream.
540	WriteSingle	Write a single-type real to the stream.
542	WriteString	Write a string to the stream.
543	WriteUnicodeString	Write a Unicode string to the stream.
543	WriteVariant	Write a variant to the stream.
540	WriteWideChar	Write widechar to stream.
543	WriteWideString	Write a widestring value to the stream.

4.80.3 Property overview

Page	Properties	Access	Description
544	Driver	r	Driver used when writing to the stream.
543	OnFindAncestor	rw	Event occurring when an ancestor component must be found.
544	OnWriteMethodProperty	rw	Handler from writing method properties.
544	OnWriteStringProperty	rw	Event handler for translating strings written to stream.
544	PropertyPath	r	Path to the property that is currently being written.
543	RootAncestor	rw	Ancestor of root component.

4.80.4 TWriter.Create

Synopsis: Creates a new Writer with a stream and bufsize.

Declaration: `constructor Create(ADriver: TAbstractObjectWriter)`
`constructor Create(Stream: TStream; BufSize: Integer)`

Visibility: public

Description: Creates a new Writer with a stream and bufsize.

4.80.5 TWriter.Destroy

Synopsis: Destroys the writer instance.

Declaration: `destructor Destroy; Override`

Visibility: public

Description: Destroys the writer instance.

4.80.6 TWriter.FlushBuffer

Synopsis: Flush the buffer

Declaration: `procedure FlushBuffer; Override`

Visibility: public

Description: `FlushBuffer` flushes the buffer. It is provided for Delphi compatibility, and is not used in FPC.

See also: `TFile.FlushBuffer` ([404](#)), `TBinaryObjectWriter.FlushBuffer` ([364](#)), `TAbstractObjectWriter.FlushBuffer` ([346](#))

4.80.7 TWriter.DefineProperty

Synopsis: Callback used when defining and streaming custom properties.

Declaration: `procedure DefineProperty(const Name: string; ReadData: TReaderProc;`
`AWriteData: TWriterProc; HasData: Boolean)`
`; Override`

Visibility: public

Description: Callback used when defining and streaming custom properties.

4.80.8 TWriter.DefineBinaryProperty

Synopsis: Callback used when defining and streaming custom properties.

Declaration: `procedure DefineBinaryProperty(const Name: string;
ReadData: TStreamProc;
AWriteData: TStreamProc; HasData: Boolean)
; Override`

Visibility: public

Description: Callback used when defining and streaming custom properties.

4.80.9 TWriter.Write

Synopsis: Write raw data to stream.

Declaration: `procedure Write(const Buffer; Count: LongInt); Virtual`

Visibility: public

Description: `Write` is introduced for Delphi compatibility to write raw data to the component stream. This should not be used in new production code as it will totally mess up the streaming.

See also: `TBinaryObjectWriter.Write` ([365](#)), `TAbstractObjectWriter.Write` ([346](#))

4.80.10 TWriter.WriteBoolean

Synopsis: Write boolean value to the stream.

Declaration: `procedure WriteBoolean(Value: Boolean)`

Visibility: public

Description: Write boolean value to the stream.

4.80.11 TWriter.WriteCollection

Synopsis: Write a collection to the stream.

Declaration: `procedure WriteCollection(Value: TCollection)`

Visibility: public

Description: Write a collection to the stream.

4.80.12 TWriter.WriteComponent

Synopsis: Stream a component to the stream.

Declaration: `procedure WriteComponent(Component: TComponent)`

Visibility: public

Description: Stream a component to the stream.

4.80.13 TWriter.WriteChar

Synopsis: Write a character to the stream.

Declaration: `procedure WriteChar(Value: Char)`

Visibility: `public`

Description: Write a character to the stream.

4.80.14 TWriter.WriteWideChar

Synopsis: Write widechar to stream.

Declaration: `procedure WriteWideChar(Value: WideChar)`

Visibility: `public`

Description: `WriteWideChar` writes a widechar to the stream. This actually writes a widestring of length 1.

See also: `TReader.ReadWideChar` ([456](#)), `TWriter.WriteString` ([543](#))

4.80.15 TWriter.WriteDescendent

Synopsis: Write descendent to stream.

Declaration: `procedure WriteDescendent(ARoot: TComponent; AAncestor: TComponent)`

Visibility: `public`

Description: `WriteDescendent` writes `ARoot` as a descendent of `AAncestor`. This is used to create diff streams: only the properties where `ARoot` differs from `AAncestor` are written to the stream.

See also: `TStream.WriteDescendent` ([471](#))

4.80.16 TWriter.WriteFloat

Synopsis: Write a float to the stream.

Declaration: `procedure WriteFloat(const Value: Extended)`

Visibility: `public`

Description: Write a float to the stream.

4.80.17 TWriter.WriteSingle

Synopsis: Write a single-type real to the stream.

Declaration: `procedure WriteSingle(const Value: Single)`

Visibility: `public`

Description: Write a single-type real to the stream.

4.80.18 TWriter.WriteDate

Synopsis: Write a date to the stream.

Declaration: `procedure WriteDate(const Value: TDateTime)`

Visibility: `public`

Description: Write a date to the stream.

4.80.19 TWriter.WriteCurrency

Synopsis: Write a currency value to the stream.

Declaration: `procedure WriteCurrency(const Value: Currency)`

Visibility: `public`

Description: `WriteCurrency` writes a currency typed value to the stream. This method does nothing except call the driver method of the driver being used.

See also: `TReader.ReadCurrency` ([457](#))

4.80.20 TWriter.WriteIdent

Synopsis: Write an identifier to the stream.

Declaration: `procedure WriteIdent(const Ident: string)`

Visibility: `public`

Description: Write an identifier to the stream.

4.80.21 TWriter.WriteInteger

Synopsis: Write an integer to the stream.

Declaration: `procedure WriteInteger(Value: LongInt); Overload`
`procedure WriteInteger(Value: Int64); Overload`

Visibility: `public`

Description: Write an integer to the stream.

4.80.22 TWriter.WriteSet

Synopsis: Write a set value to the stream.

Declaration: `procedure WriteSet(Value: LongInt; SetType: Pointer)`

Visibility: `public`

Description: `WriteSet` writes a set `Value` consisting of elements with type `EnumType`. The set must be encoded as an integer where each element is encoded in a bit of the integer. Thus, at most an enumerated type with 32 elements can be written with this method.

Errors: No checking is performed on the validity of `EnumType`. It is assumed to be a valid `PTypeInfo` pointer.

See also: `TReader.ReadSet` ([458](#))

4.80.23 TWriter.WriteListBegin

Synopsis: Write a start-of-list marker to the stream.

Declaration: `procedure WriteListBegin`

Visibility: `public`

Description: Write a start-of-list marker to the stream.

4.80.24 TWriter.WriteListEnd

Synopsis: Write an end-of-list marker to the stream.

Declaration: `procedure WriteListEnd`

Visibility: `public`

Description: Write an end-of-list marker to the stream.

4.80.25 TWriter.WriteSignature

Synopsis: Write a signature to the stream.

Declaration: `procedure WriteSignature`

Visibility: `public`

Description: `WriteSignature` writes the streaming signature (if any) to a stream. It is called once, at the start of writing the root component to a stream.

See also: `TAbstractObjectWriter.WriteSignature` ([345](#)), `TBinaryObjectWriter.WriteSignature` ([364](#))

4.80.26 TWriter.WriteRootComponent

Synopsis: Write a root component to the stream.

Declaration: `procedure WriteRootComponent (ARoot: TComponent)`

Visibility: `public`

Description: Write a root component to the stream.

4.80.27 TWriter.WriteString

Synopsis: Write a string to the stream.

Declaration: `procedure WriteString(const Value: string)`

Visibility: `public`

Description: Write a string to the stream.

4.80.28 TWriter.WriteString

Synopsis: Write a widestring value to the stream.

Declaration: `procedure WriteWideString(const Value: WideString)`

Visibility: public

Description: `WriteWideString` writes a currency typed value to the stream. This method does nothing except call the driver method of the driver being used.

See also: `TReader.ReadWideString` ([460](#))

4.80.29 TWriter.WriteString

Synopsis: Write a Unicode string to the stream.

Declaration: `procedure WriteUnicodeString(const Value: UnicodeString)`

Visibility: public

Description: `WriteUnicodeString` writes `Value`, a `UnicodeString` string to the stream. It simply passes the string on to the `WriteUnicodeString` method of the writer driver class.

See also: `TBinaryObjectWriter.WriteString` ([367](#)), `TReader.ReadUnicodeString` ([460](#))

4.80.30 TWriter.WriteVariant

Synopsis: Write a variant to the stream.

Declaration: `procedure WriteVariant(const VarValue: Variant)`

Visibility: public

Description: `WriteVariant` writes `Value`, a simple variant, to the stream. It simply passes the string on to the `WriteVariant` method of the writer driver class.

See also: `TBinaryObjectWriter.WriteVariant` ([368](#)), `TReader.ReadVariant` ([459](#))

4.80.31 TWriter.RootAncestor

Synopsis: Ancestor of root component.

Declaration: `Property RootAncestor : TComponent`

Visibility: public

Access: Read,Write

Description: Ancestor of root component.

4.80.32 TWriter.OnFindAncestor

Synopsis: Event occurring when an ancestor component must be found.

Declaration: `Property OnFindAncestor : TFindAncestorEvent`

Visibility: public

Access: Read,Write

Description: Event occurring when an ancestor component must be found.

4.80.33 TWriter.OnWriteMethodProperty

Synopsis: Handler from writing method properties.

Declaration: `Property OnWriteMethodProperty : TWriteMethodPropertyEvent`

Visibility: `public`

Access: `Read, Write`

Description: `OnWriteMethodProperty` can be set by an IDE or some streaming mechanism which handles dummy values for method properties; It can be used to write a real value to the stream which will be interpreted correctly when the stream is read. See `TWriteMethodPropertyEvent` (302) for a description of the arguments.

See also: `TWriteMethodPropertyEvent` (302), `TReader.OnSetMethodProperty` (462)

4.80.34 TWriter.OnWriteStringProperty

Synopsis: Event handler for translating strings written to stream.

Declaration: `Property OnWriteStringProperty : TReadWriteStringPropertyEvent`

Visibility: `public`

Access: `Read, Write`

Description: `OnWriteStringProperty` is called whenever a string property is written to the stream. It can be used e.g. by a translation mechanism to translate the strings on the fly, when a form is written. See `TReadWriteStringPropertyEvent` (294) for a description of the various parameters.

See also: `TReader.OnPropertyNotFound` (461), `TReader.OnSetMethodProperty` (462), `TReadWriteStringPropertyEvent` (294)

4.80.35 TWriter.Driver

Synopsis: Driver used when writing to the stream.

Declaration: `Property Driver : TAbstractObjectWriter`

Visibility: `public`

Access: `Read`

Description: Driver used when writing to the stream.

4.80.36 TWriter.PropertyPath

Synopsis: Path to the property that is currently being written.

Declaration: `Property PropertyPath : string`

Visibility: `public`

Access: `Read`

Description: `PropertyPath` is set to the property name of the class currently being written to stream. This is only done when `TPersistent` (446) descendant class properties are written.

Chapter 5

Reference for unit 'clocale'

5.1 Used units

Table 5.1: Used units by unit 'clocale'

Name	Page
System	1335

5.2 Overview

The `clocale` offers no API by itself: it just initializes the internationalization settings of the `sysutils` ([1609](#)) unit with the values provided by the C library found on most Unix or Linux systems that are POSIX compliant.

The `clocale` should simply be included in the `uses` clause of the program, preferably as one of the first units, and the initialization section of the unit will do all the work.

Note that including this unit, links your program to the C library of the system.

It makes no sense to use this unit on a non-POSIX system: Windows, OS/2 or DOS - therefore it should always be between an `ifdef` statement:

```
program myprogram;

uses
  {$ifdef unix}clocale{$endif},
  classes, sysutils;
```

Chapter 6

Reference for unit 'cmem'

6.1 Used units

Table 6.1: Used units by unit 'cmem'

Name	Page
System	1335

6.2 Overview

The `cmem` memory manager sets the system units memory manager to a C-based memory manager: all memory management calls are shunted through to the C memory manager, using `Malloc` ([547](#)), `Free` ([547](#)) and `ReAlloc` ([547](#)). For this reason, the `cmem` unit should be the first unit of the uses clause of the program.

The unit also offers the C memory calls directly as external declarations from the C library, but it is recommended to use the normal FPC routines for this.

Obviously, including this unit links your program to the C library.

Remark Note that specifying the `-gv` command-line option, to enable valgrind debugging info will implicitly add this unit to your program.

6.3 Constants, types and variables

6.3.1 Constants

`LibName = 'c'`

`LibName` is the name of the library that is actually used. On most systems, this is simply "libc.so".

6.4 Procedures and functions

6.4.1 CAlloc

Synopsis: Allocate memory based on item size and count.

Declaration: `function CAlloc(unitSize: PtrUInt; UnitCount: PtrUInt) : pointer`

Visibility: `default`

Description: `CAlloc` allocates memory to hold `UnitCount` units of size `UnitSize` each. The memory is one block of memory. It returns a pointer to the newly allocated memory block.

See also: `Malloc` ([547](#)), `Free` ([547](#)), `Realloc` ([547](#))

6.4.2 Free

Synopsis: Free a previously allocated block.

Declaration: `procedure Free(P: pointer)`

Visibility: `default`

Description: `Free` returns the memory block pointed to by `P` to the system. After `Free` was called, the pointer `P` is no longer valid.

See also: `Malloc` ([547](#)), `ReAlloc` ([547](#))

6.4.3 Malloc

Synopsis: `Malloc` external declaration.

Declaration: `function Malloc(Size: PtrUInt) : Pointer`

Visibility: `default`

Description: `Malloc` is the external declaration of the C libraries `malloc` call. It accepts a size parameter, and returns a pointer to a memory block of the requested size or `Nil` if no more memory could be allocated.

See also: `Free` ([547](#)), `ReAlloc` ([547](#))

6.4.4 ReAlloc

Synopsis: Reallocates a memory block.

Declaration: `function ReAlloc(P: Pointer; Size: PtrUInt) : pointer`

Visibility: `default`

Description: `ReAlloc` re-allocates a block of memory pointed to by `p`. The new block will have size `Size`, and as much data as was available or as much data as fits is copied from the old to the new location.

See also: `Malloc` ([547](#)), `Free` ([547](#))

Chapter 7

Reference for unit 'collation_de'

7.1 Used units

Table 7.1: Used units by unit 'collation_de'

Name	Page
System	1335

7.2 Overview

The `collation_de` unit registers the German Unicode collation (de). This collation bases itself on the DUCET collation, so that collation will be included as well.

This unit does not contain any routines. It simply registers the collation in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 8

Reference for unit 'collation_es'

8.1 Used units

Table 8.1: Used units by unit 'collation_es'

Name	Page
System	1335

8.2 Overview

The `collation_es` unit registers the Spanish Unicode collation (de). This collation bases itself on the DUCET collation, so that collation will be included as well.

This unit does not contain any routines. It simply registers the collation in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 9

Reference for unit 'collation_fr_ca'

9.1 Used units

Table 9.1: Used units by unit 'collation_fr_ca'

Name	Page
System	1335

9.2 Overview

The `collation_fr_ca` unit registers the French Unicode collation (fr). This collation bases itself on the DUCET collation, so that collation will be included as well.

This unit does not contain any routines. It simply registers the collation in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 10

Reference for unit 'collation_ja'

10.1 Used units

Table 10.1: Used units by unit 'collation_ja'

Name	Page
System	1335

10.2 Overview

The `collation_ja` unit registers the Japanese Unicode collation (ja). This collation bases itself on the DUCET collation, so that collation will be included as well.

This unit does not contain any routines. It simply registers the collation in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 11

Reference for unit 'collation_ko'

11.1 Used units

Table 11.1: Used units by unit 'collation_ko'

Name	Page
System	1335

11.2 Overview

The `collation_ko` unit registers the Korean Unicode collation (ko). This collation bases itself on the DUCET collation, so that collation will be included as well.

This unit does not contain any routines. It simply registers the collation in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 12

Reference for unit 'collation_ru'

12.1 Used units

Table 12.1: Used units by unit 'collation_ru'

Name	Page
System	1335

12.2 Overview

The `collation_ru` unit registers the Russian Unicode collation (ru). This collation bases itself on the DUCET collation, so that collation will be included as well.

This unit does not contain any routines. It simply registers the collation in the initialization section of the unit, so including the unit in the `uses` clause of the program is sufficient.

Chapter 13

Reference for unit 'collation_sv'

13.1 Used units

Table 13.1: Used units by unit 'collation_sv'

Name	Page
System	1335

13.2 Overview

The `collation_sv` unit registers the Swedish Unicode collation (sv). This collation bases itself on the DUCET collation, so that collation will be included as well.

This unit does not contain any routines. It simply registers the collation in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 14

Reference for unit 'collation_zh'

14.1 Used units

Table 14.1: Used units by unit 'collation_zh'

Name	Page
System	1335

14.2 Overview

The `collation_zh` unit registers the Chinese Unicode collation (zh). This collation bases itself on the DUCET collation, so that collation will be included as well.

This unit does not contain any routines. It simply registers the collation in the initialization section of the unit, so including the unit in the uses clause of the program is sufficient.

Chapter 15

Reference for unit 'cp1250'

15.1 Used units

Table 15.1: Used units by unit 'cp1250'

Name	Page
System	1335

15.2 Overview

The `cp1250` unit registers single-byte codepage 1250. This is necessary to convert single-byte strings using codepage 1250 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the `uses` clause of the program is sufficient.

Chapter 16

Reference for unit 'cp1251'

16.1 Used units

Table 16.1: Used units by unit 'cp1251'

Name	Page
System	1335

16.2 Overview

The `cp1251` unit registers single-byte codepage 1251. This is necessary to convert single-byte strings using codepage 1251 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the `uses` clause of the program is sufficient.

Chapter 17

Reference for unit 'cp1252'

17.1 Used units

Table 17.1: Used units by unit 'cp1252'

Name	Page
System	1335

17.2 Overview

The `cp1252` unit registers single-byte codepage 1252. This is necessary to convert single-byte strings using codepage 1252 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the `uses` clause of the program is sufficient.

Chapter 18

Reference for unit 'cp1253'

18.1 Used units

Table 18.1: Used units by unit 'cp1253'

Name	Page
System	1335

18.2 Overview

The `cp1253` unit registers single-byte codepage 1253. This is necessary to convert single-byte strings using codepage 1253 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the `uses` clause of the program is sufficient.

Chapter 19

Reference for unit 'cp1254'

19.1 Used units

Table 19.1: Used units by unit 'cp1254'

Name	Page
System	1335

19.2 Overview

The `cp1254` unit registers single-byte codepage 1254. This is necessary to convert single-byte strings using codepage 1254 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the `uses` clause of the program is sufficient.

Chapter 20

Reference for unit 'cp1255'

20.1 Used units

Table 20.1: Used units by unit 'cp1255'

Name	Page
System	1335

20.2 Overview

The `cp1255` unit registers single-byte codepage 1255. This is necessary to convert single-byte strings using codepage 1255 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the `uses` clause of the program is sufficient.

Chapter 21

Reference for unit 'cp1256'

21.1 Used units

Table 21.1: Used units by unit 'cp1256'

Name	Page
System	1335

21.2 Overview

The `cp1256` unit registers single-byte codepage 1256. This is necessary to convert single-byte strings using codepage 1256 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the `uses` clause of the program is sufficient.

Chapter 22

Reference for unit 'cp1257'

22.1 Used units

Table 22.1: Used units by unit 'cp1257'

Name	Page
System	1335

22.2 Overview

The `cp1257` unit registers single-byte codepage 1257. This is necessary to convert single-byte strings using codepage 1257 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the `uses` clause of the program is sufficient.

Chapter 23

Reference for unit 'cp1258'

23.1 Used units

Table 23.1: Used units by unit 'cp1258'

Name	Page
System	1335

23.2 Overview

The `cp1258` unit registers single-byte codepage 1258. This is necessary to convert single-byte strings using codepage 1258 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the `uses` clause of the program is sufficient.

Chapter 24

Reference for unit 'cp437'

24.1 Used units

Table 24.1: Used units by unit 'cp437'

Name	Page
System	1335

24.2 Overview

The `cp437` unit registers single-byte codepage 437. This is necessary to convert single-byte strings using codepage 437 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the `uses` clause of the program is sufficient.

Chapter 25

Reference for unit 'cp646'

25.1 Used units

Table 25.1: Used units by unit 'cp646'

Name	Page
System	1335

25.2 Overview

The `cp646` unit registers single-byte codepage 646. This is necessary to convert single-byte strings using codepage 646 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the `uses` clause of the program is sufficient.

Chapter 26

Reference for unit 'cp850'

26.1 Used units

Table 26.1: Used units by unit 'cp850'

Name	Page
System	1335

26.2 Overview

The `cp850` unit registers single-byte codepage 850. This is necessary to convert single-byte strings using codepage 850 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the `uses` clause of the program is sufficient.

Chapter 27

Reference for unit 'cp852'

27.1 Used units

Table 27.1: Used units by unit 'cp852'

Name	Page
System	1335

27.2 Overview

The `cp852` unit registers single-byte codepage 852. This is necessary to convert single-byte strings using codepage 852 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the `uses` clause of the program is sufficient.

Chapter 28

Reference for unit 'cp856'

28.1 Used units

Table 28.1: Used units by unit 'cp856'

Name	Page
System	1335

28.2 Overview

The `cp856` unit registers single-byte codepage 856. This is necessary to convert single-byte strings using codepage 856 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the `uses` clause of the program is sufficient.

Chapter 29

Reference for unit 'cp866'

29.1 Used units

Table 29.1: Used units by unit 'cp866'

Name	Page
System	1335

29.2 Overview

The `cp866` unit registers single-byte codepage 866. This is necessary to convert single-byte strings using codepage 866 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the `uses` clause of the program is sufficient.

Chapter 30

Reference for unit 'cp874'

30.1 Used units

Table 30.1: Used units by unit 'cp874'

Name	Page
System	1335

30.2 Overview

The `cp874` unit registers single-byte codepage 874. This is necessary to convert single-byte strings using codepage 874 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the `uses` clause of the program is sufficient.

Chapter 31

Reference for unit 'cp8859_1'

31.1 Used units

Table 31.1: Used units by unit 'cp8859_1'

Name	Page
System	1335

31.2 Overview

The `cp8859_1` unit registers single-byte codepage 8859-1. This is necessary to convert single-byte strings using codepage 8859-1 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the `uses` clause of the program is sufficient.

Chapter 32

Reference for unit 'cp8859_2'

32.1 Used units

Table 32.1: Used units by unit 'cp8859_2'

Name	Page
System	1335

32.2 Overview

The `cp8859_2` unit registers single-byte codepage 8859-2. This is necessary to convert single-byte strings using codepage 8859-2 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the `uses` clause of the program is sufficient.

Chapter 33

Reference for unit 'cp8859_5'

33.1 Used units

Table 33.1: Used units by unit 'cp8859_5'

Name	Page
System	1335

33.2 Overview

The `cp8859_5` unit registers single-byte codepage 8859-5. This is necessary to convert single-byte strings using codepage 8859-5 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the `uses` clause of the program is sufficient.

Chapter 34

Reference for unit 'cp895'

34.1 Used units

Table 34.1: Used units by unit 'cp895'

Name	Page
System	1335

34.2 Overview

The `cp895` unit registers single-byte codepage 895. This is necessary to convert single-byte strings using codepage 895 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the `uses` clause of the program is sufficient.

Chapter 35

Reference for unit 'cp932'

35.1 Used units

Table 35.1: Used units by unit 'cp932'

Name	Page
System	1335

35.2 Overview

The `cp932` unit registers single-byte codepage 932. This is necessary to convert single-byte strings using codepage 932 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the `uses` clause of the program is sufficient.

Chapter 36

Reference for unit 'cp936'

36.1 Used units

Table 36.1: Used units by unit 'cp936'

Name	Page
System	1335

36.2 Overview

The `cp936` unit registers single-byte codepage 936. This is necessary to convert single-byte strings using codepage 936 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the `uses` clause of the program is sufficient.

Chapter 37

Reference for unit 'cp949'

37.1 Used units

Table 37.1: Used units by unit 'cp949'

Name	Page
System	1335

37.2 Overview

The `cp949` unit registers single-byte codepage 949. This is necessary to convert single-byte strings using codepage 949 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the `uses` clause of the program is sufficient.

Chapter 38

Reference for unit 'cp950'

38.1 Used units

Table 38.1: Used units by unit 'cp950'

Name	Page
System	1335

38.2 Overview

The `cp950` unit registers single-byte codepage 950. This is necessary to convert single-byte strings using codepage 950 to unicode strings.

This unit does not contain any routines. It simply registers the code page data in the initialization section of the unit, so including the unit in the `uses` clause of the program is sufficient.

Chapter 39

Reference for unit 'cpall'

39.1 Used units

Table 39.1: Used units by unit 'cpall'

Name	Page
cp1250	556
cp1251	557
cp1252	558
cp1253	559
cp1254	560
cp1255	561
cp1256	562
cp1257	563
cp1258	564
cp3021	??
cp437	565
cp646	566
cp737	??
cp775	??
cp850	567
cp852	568
cp855	??
cp856	569
cp857	??
cp860	??
cp861	??
cp862	??
cp863	??
cp864	??
cp865	??
cp866	570
cp869	??
cp874	571
cp8859_1	572
cp8859_10	??
cp8859_11	??
cp8859_13	??
cp8859_14	??
cp8859_15	??
cp8859_16	??
cp8859_2	573
cp8859_3	??
cp8859_4	??
cp8859_5	574

39.2 Overview

The `cpall` unit registers all known single-byte codepages: 1251 ([581](#)), 866 ([581](#)), ISO 8856-5 ([581](#)) (cyrillic), 8859-1 ([581](#)), 8859-2 ([581](#)), 1253 ([581](#)) (greek), 850 ([581](#)), 437 ([581](#)), 1252 ([581](#)), 646 ([581](#)), 874 ([581](#)), 856 ([581](#)), 1250 ([581](#)), 1254 ([581](#)), 1255 ([581](#)), 1256 ([581](#)), 1257 ([581](#)), 1258 ([581](#)), 852 ([581](#)).

This unit does not contain any routines. It simply uses the other units so all corresponding code pages are registered.

Chapter 40

Reference for unit 'Crt'

40.1 Used units

Table 40.1: Used units by unit 'Crt'

Name	Page
System	1335

40.2 Overview

This chapter describes the CRT unit for Free Pascal, under all of Dos, Linux and Windows. The unit was first written for Dos by Florian Klaempfl. The unit was ported to Linux by Mark May and enhanced by Michael Van Canneyt and Peter Vreman. It works on the Linux console, and in xterm and rxvt windows under X-Windows. The functionality for both is the same, except that under Linux the use of an early implementation (versions 0.9.1 and earlier of the compiler) the CRT unit automatically cleared the screen at program startup.

There are some caveats when using the CRT unit:

- Programs using the CRT unit will *not* be usable when input/output is being redirected on the command-line.
- For similar reasons they are not usable as CGI-scripts for use with a webserver.
- The use of the CRT unit and the graph unit may not always be supported.
- The CRT unit is not thread safe.
- On Linux or other Unix OSes, executing other programs that expect special terminal behaviour (using one of the special functions in the Linux unit) will not work. The terminal is set in RAW mode, which will destroy most terminal emulation settings.
- The CRT unit stems from the TP/Dos area. It is designed to work with single-byte character sets, where 1 char = 1 byte. That means that widestrings or UTF-8 encoded (ansi)strings will not work correctly.

40.3 Constants, types and variables

40.3.1 Constants

`Black = 0`

Black color attribute.

`Blink = 128`

Blink attribute.

`Blue = 1`

Blue color attribute.

`Brown = 6`

Brown color attribute.

`BW40 = 0`

40 columns black and white screen mode.

`BW80 = 2`

80 columns black and white screen mode.

`C40 = CO40`

40 columns color screen mode.

`C80 = CO80`

80 columns color screen mode.

`CO40 = 1`

40 columns color screen mode.

`CO80 = 3`

80 columns color screen mode.

`ConsoleMaxX = 1024`

`ConsoleMaxY = 1024`

`Cyan = 3`

Cyan color attribute.

DarkGray = 8

Dark gray color attribute.

Flushing = False

Font8x8 = 256

Internal ROM font mode.

Green = 2

Green color attribute.

LightBlue = 9

Light Blue color attribute.

LightCyan = 11

Light cyan color attribute.

LightGray = 7

Light gray color attribute.

LightGreen = 10

Light green color attribute.

LightMagenta = 13

Light magenta color attribute.

LightRed = 12

Light red color attribute.

Magenta = 5

Magenta color attribute.

Mono = 7

Monochrome screen mode (hercules screens).

Red = 4

Red color attribute.

```
ScreenHeight : LongInt = 25
```

Current screen height.

```
ScreenWidth : LongInt = 80
```

Current screen width.

```
White = 15
```

White color attribute.

```
Yellow = 14
```

Yellow color attribute.

40.3.2 Types

```
tcrtcoord = 1..255
```

`tcrtcoord` is a subrange type for denoting CRT coordinates. It supports coordinates ranging from 1 to 255. Using this type together with range-checking turned on can be used to debug CRT code.

40.3.3 Variables

```
CheckBreak : Boolean
```

Check for CTRL-Break keystroke. Not used.

```
CheckEOF : Boolean
```

Check for EOF on standard input. Not used.

```
CheckSnow : Boolean
```

Check snow on CGA screens. Not used.

```
ConsoleBuf : PConsoleBuf
```

```
DirectVideo : Boolean
```

The `DirectVideo` variable controls the writing to the screen. If it is `True`, the cursor is set via direct port access. If `False`, then the BIOS is used. This is defined under dos only.

```
LastMode : Word = 3
```

The `Lastmode` variable tells you which mode was last selected for the screen. It is defined on DOS only.

`TextAttr : Byte = $07`

The `TextAttr` variable controls the attributes with which characters are written to screen.

`WindMax : Word = $184f`

The upper byte of `WindMax` contains the Y coordinate while the lower byte contains the X coordinate. The use of this variable is deprecated, use `WindMaxX` and `WindMaxY` instead.

`WindMaxX : DWord`

X coordinate of lower right corner of the defined window.

`WindMaxY : DWord`

Y coordinate of lower right corner of the defined window.

`WindMin : Word = $0`

The upper byte of `WindMin` contains the Y coordinate while the lower byte contains the X coordinate. The use of this variable is deprecated, use `WindMinX` and `WindMinY` instead.

`WindMinX : DWord`

X coordinate of upper left corner of the defined window.

`WindMinY : DWord`

Y coordinate of upper left corner of the defined window.

40.4 Procedures and functions

40.4.1 AssignCrt

Synopsis: Assign file to CRT.

Declaration: `procedure AssignCrt (var F: Text)`

Visibility: default

Description: `AssignCrt` Assigns a file `F` to the console. Everything written to the file `F` goes to the console instead. If the console contains a window, everything is written to the window instead.

Errors: None.

See also: `Window` ([598](#))

Listing: `./examples/crtex/ex1.pp`

```

Program Example1;
uses Crt;

{ Program to demonstrate the AssignCrt function. }

var
  F : Text;
begin
  AssignCrt(F);
  Rewrite(F); { Don't forget to open for output! }
  WriteLn(F,'This is written to the Assigned File');
  Close(F);
end.

```

40.4.2 ClrEol

Synopsis: Clear from cursor position till end of line.

Declaration: `procedure ClrEol`

Visibility: default

Description: `ClrEol` clears the current line, starting from the cursor position, to the end of the window. The cursor doesn't move

Errors: None.

See also: `DelLine` ([590](#)), `InsLine` ([592](#)), `ClrScr` ([589](#))

Listing: `./examples/crtex/ex9.pp`

```

Program Example9;
uses Crt;

{ Program to demonstrate the ClrEol function. }
var
  I,J : integer;

begin
  For I:=1 to 15 do
    For J:=1 to 80 do
      begin
        gotoxy(j,i);
        Write(j mod 10);
      end;
  Window(5,5,75,12);
  Write('This line will be cleared from',
        ' here till the right of the window');
  GotoXY(27,WhereY);
  ReadKey;
  ClrEol;
  WriteLn;
end.

```

40.4.3 ClrScr

Synopsis: Clear current window.

Declaration: `procedure ClrScr`

Visibility: default

Description: `ClrScr` clears the current window (using the current colors), and sets the cursor in the top left corner of the current window.

Errors: None.

See also: Window ([598](#))

Listing: `./examples/crtex/ex8.pp`

```
Program Example8;
uses Crt;

{ Program to demonstrate the ClrScr function. }

begin
  Writeln('Press any key to clear the screen');
  ReadKey;
  ClrScr;
  Writeln('Have fun with the cleared screen');
end.
```

40.4.4 cursorbig

Synopsis: Show big cursor.

Declaration: `procedure cursorbig`

Visibility: default

Description: `CursorBig` makes the cursor a big rectangle. Not implemented on unixes.

Errors: None.

See also: `CursorOn` ([590](#)), `CursorOff` ([589](#))

40.4.5 cursoroff

Synopsis: Hide cursor.

Declaration: `procedure cursoroff`

Visibility: default

Description: `CursorOff` switches the cursor off (i.e. the cursor is no longer visible). Not implemented on unixes.

Errors: None.

See also: `CursorOn` ([590](#)), `CursorBig` ([589](#))

40.4.6 cursoron

Synopsis: Display cursor.

Declaration: `procedure cursoron`

Visibility: default

Description: `CursorOn` switches the cursor on. Not implemented on unixes.

Errors: None.

See also: `CursorBig` ([589](#)), `CursorOff` ([589](#))

40.4.7 Delay

Synopsis: Delay program execution.

Declaration: `procedure Delay (MS: Word)`

Visibility: default

Description: `Delay` waits a specified number of milliseconds. The number of specified seconds is an approximation, and may be off a lot, if system load is high.

Errors: None

See also: `Sound` ([595](#)), `NoSound` ([594](#))

Listing: `./examples/crtex/ex15.pp`

```

Program Example15;
uses Crt;

{ Program to demonstrate the Delay function. }
var
  i : longint;
begin
  WriteLn('Counting Down');
  for i:=10 downto 1 do
    begin
      WriteLn(i);
      Delay(1000); {Wait one second}
    end;
  WriteLn('BOOM!!!!');
end.

```

40.4.8 DelLine

Synopsis: Delete line at cursor position.

Declaration: `procedure DelLine`

Visibility: default

Description: `DelLine` removes the current line. Lines following the current line are scrolled 1 line up, and an empty line is inserted at the bottom of the current window. The cursor doesn't move.

Errors: None.

See also: [ClrEol \(588\)](#), [InsLine \(592\)](#), [ClrScr \(589\)](#)

Listing: ./examples/crtex/ex11.pp

```

Program Example10;
uses Crt;

{ Program to demonstrate the InsLine function. }

begin
  ClrScr;
  WriteLn;
  WriteLn('Line 1');
  WriteLn('Line 2');
  WriteLn('Line 2');
  WriteLn('Line 3');
  WriteLn;
  WriteLn('Oops, Line 2 is listed twice,',
    ' let"s delete the line at the cursor postion');
  GotoXY(1,3);
  ReadKey;
  DelLine;
  GotoXY(1,10);
end.

```

40.4.9 GotoXY

Synopsis: Set cursor position on screen.

Declaration: `procedure GotoXY(X: tcoord; Y: tcoord)`

Visibility: default

Description: `GotoXY` positions the cursor at (X, Y), X in horizontal, Y in vertical direction relative to the origin of the current window. The origin is located at (1, 1), the upper-left corner of the window.

Errors: None.

See also: [WhereX \(597\)](#), [WhereY \(597\)](#), [Window \(598\)](#)

Listing: ./examples/crtex/ex6.pp

```

Program Example6;
uses Crt;

{ Program to demonstrate the GotoXY function. }

begin
  ClrScr;
  GotoXY(10,10);
  Write('10,10');
  GotoXY(70,20);
  Write('70,20');
  GotoXY(1,22);
end.

```

40.4.10 HighVideo

Synopsis: Switch to highlighted text mode.

Declaration: `procedure HighVideo`

Visibility: `default`

Description: `HighVideo` switches the output to highlighted text. (It sets the high intensity bit of the video attribute)

Errors: None.

See also: `TextColor` ([596](#)), `TextBackground` ([595](#)), `LowVideo` ([593](#)), `NormVideo` ([594](#))

Listing: `./examples/crtex/ex14.pp`

```
Program Example14;
uses Crt;

{ Program to demonstrate the LowVideo, HighVideo, NormVideo functions. }

begin
  LowVideo;
  WriteLn('This is written with LowVideo');
  HighVideo;
  WriteLn('This is written with HighVideo');
  NormVideo;
  WriteLn('This is written with NormVideo');
end.
```

40.4.11 InsLine

Synopsis: Insert an empty line at cursor position.

Declaration: `procedure InsLine`

Visibility: `default`

Description: `InsLine` inserts an empty line at the current cursor position. Lines following the current line are scrolled 1 line down, causing the last line to disappear from the window. The cursor doesn't move.

Errors: None.

See also: `ClrEol` ([588](#)), `DelLine` ([590](#)), `ClrScr` ([589](#))

Listing: `./examples/crtex/ex10.pp`

```
Program Example10;
uses Crt;

{ Program to demonstrate the InsLine function. }

begin
  ClrScr;
  WriteLn;
  WriteLn('Line 1');
  WriteLn('Line 3');
  WriteLn;
```

```

WriteLn('Oops, forgot Line 2, let"s insert at the cursor postion');
GotoXY(1,3);
ReadKey;
InsLine;
Write('Line 2');
GotoXY(1,10);
end.

```

40.4.12 KeyPressed

Synopsis: Check if there is a keypress in the keybuffer.

Declaration: `function KeyPressed : Boolean`

Visibility: default

Description: `KeyPressed` scans the keyboard buffer and sees if a key has been pressed. If this is the case, `True` is returned. If not, `False` is returned. The `Shift`, `Alt`, `Ctrl` keys are not reported. The key is not removed from the buffer, and can hence still be read after the `KeyPressed` function has been called.

Errors: None.

See also: `ReadKey` ([594](#))

Listing: `./examples/crtex/ex2.pp`

```

Program Example2;
uses Crt;

{ Program to demonstrate the KeyPressed function. }

begin
  WriteLn('Waiting until a key is pressed');
  repeat
    until KeyPressed;
  { The key is not Read,
    so it should also be outputted at the commandline }
end.

```

40.4.13 LowVideo

Synopsis: Switch to low intensity colors.

Declaration: `procedure LowVideo`

Visibility: default

Description: `LowVideo` switches the output to non-highlighted text. (It clears the high intensity bit of the video attribute)

For an example, see `HighVideo` ([592](#))

Errors: None.

See also: `TextColor` ([596](#)), `TextBackground` ([595](#)), `HighVideo` ([592](#)), `NormVideo` ([594](#))

40.4.14 NormVideo

Synopsis: Return to normal (startup) modus.

Declaration: `procedure NormVideo`

Visibility: default

Description: `NormVideo` switches the output to the defaults, read at startup. (The defaults are read from the cursor position at startup)

For an example, see `HighVideo` ([592](#))

Errors: None.

See also: `TextColor` ([596](#)), `TextBackground` ([595](#)), `LowVideo` ([593](#)), `HighVideo` ([592](#))

40.4.15 NoSound

Synopsis: Stop system speaker.

Declaration: `procedure NoSound`

Visibility: default

Description: `NoSound` stops the speaker sound. This call is not supported on all operating systems.

Errors: None.

See also: `Sound` ([595](#))

Listing: `./examples/crtex/ex16.pp`

```

Program Example16;
uses Crt;

{ Program to demonstrate the Sound and NoSound function. }

var
  i : longint;
begin
  WriteLn('You will hear some tones from your speaker');
  i:=0;
  while (i<15000) do
    begin
      inc(i,500);
      Sound(i);
      Delay(100);
    end;
  WriteLn('Quiet now!');
  NoSound; {Stop noise}
end.

```

40.4.16 ReadKey

Synopsis: Read key from keybuffer.

Declaration: `function ReadKey : Char`

Visibility: default

Description: `ReadKey` reads 1 key from the keyboard buffer, and returns this. If an extended or function key has been pressed, then the zero ASCII code is returned. You can then read the scan code of the key with a second `ReadKey` call.

Key mappings under Linux can cause the wrong key to be reported by `ReadKey`, so caution is needed when using `ReadKey`.

Errors: None.

See also: `KeyPressed` ([593](#))

Listing: `./examples/crtex/ex3.pp`

```

Program Example3;
uses Crt;

{ Program to demonstrate the ReadKey function. }

var
  ch : char;
begin
  writeln('Press Left/Right, Esc=Quit');
  repeat
    ch:=ReadKey;
    case ch of
      #0 : begin
        ch:=ReadKey; {Read ScanCode}
        case ch of
          #75 : WriteLn('Left');
          #77 : WriteLn('Right');
        end;
      end;
      #27 : WriteLn('ESC');
    end;
  until ch=#27 {Esc}
end.
```

40.4.17 Sound

Synopsis: Sound system speaker.

Declaration: `procedure Sound(Hz : Word)`

Visibility: default

Description: `Sound` sounds the speaker at a frequency of `hz`. Under Windows, a system sound is played and the frequency parameter is ignored. On other operating systems, this routine may not be implemented.

Errors: None.

See also: `NoSound` ([594](#))

40.4.18 TextBackground

Synopsis: Set text background.

Declaration: `procedure TextBackground(Color: Byte)`

Visibility: default

Description: `TextBackground` sets the background color to CL. CL can be one of the predefined color constants.

Errors: None.

See also: `TextColor` (596), `HighVideo` (592), `LowVideo` (593), `NormVideo` (594)

Listing: `./examples/crtex/ex13.pp`

```
Program Example13;
uses Crt;

{ Program to demonstrate the TextBackground function. }

begin
  TextColor(White);
  WriteLn('This is written in with the default background color');
  TextBackground(Green);
  WriteLn('This is written in with a Green background');
  TextBackground(Brown);
  WriteLn('This is written in with a Brown background');
  TextBackground(Black);
  WriteLn('Back with a black background');
end.
```

40.4.19 TextColor

Synopsis: Set text color.

Declaration: `procedure TextColor(Color: Byte)`

Visibility: default

Description: `TextColor` sets the foreground color to CL. CL can be one of the predefined color constants.

Errors: None.

See also: `TextBackground` (595), `HighVideo` (592), `LowVideo` (593), `NormVideo` (594)

Listing: `./examples/crtex/ex12.pp`

```
Program Example12;
uses Crt;

{ Program to demonstrate the TextColor function. }

begin
  WriteLn('This is written in the default color');
  TextColor(Red);
  WriteLn('This is written in Red');
  TextColor(White);
  WriteLn('This is written in White');
  TextColor(LightBlue);
  WriteLn('This is written in Light Blue');
end.
```

40.4.20 TextMode

Synopsis: Set screen mode.

Declaration: `procedure TextMode (Mode: Word)`

Visibility: default

Description: `TextMode` sets the textmode of the screen (i.e. the number of lines and columns of the screen). The lower byte is use to set the VGA text mode.
This procedure is only implemented on dos.

Errors: None.

See also: `Window` ([598](#))

40.4.21 WhereX

Synopsis: Return X (horizontal) cursor position.

Declaration: `function WhereX : tcrtcoord`

Visibility: default

Description: `WhereX` returns the current X-coordinate of the cursor, relative to the current window. The origin is $(1, 1)$, in the upper-left corner of the window.

Errors: None.

See also: `GotoXY` ([591](#)), `WhereY` ([597](#)), `Window` ([598](#))

Listing: `./examples/crtex/ex7.pp`

```
Program Example7;
uses Crt;

{ Program to demonstrate the WhereX and WhereY functions. }

begin
  Writeln('Cursor postion: X=',WhereX,' Y=',WhereY);
end.
```

40.4.22 WhereY

Synopsis: Return Y (vertical) cursor position.

Declaration: `function WhereY : tcrtcoord`

Visibility: default

Description: `WhereY` returns the current Y-coordinate of the cursor, relative to the current window. The origin is $(1, 1)$, in the upper-left corner of the window.

Errors: None.

See also: `GotoXY` ([591](#)), `WhereX` ([597](#)), `Window` ([598](#))

Listing: `./examples/crtex/ex7.pp`

```

Program Example7;
uses Crt;

{ Program to demonstrate the WhereX and WhereY functions. }

begin
  WriteLn('Cursor postion: X=',WhereX,' Y=',WhereY);
end.

```

40.4.23 Window

Synopsis: Create new window on screen.

Declaration: `procedure Window(X1: Byte; Y1: Byte; X2: Byte; Y2: Byte)`

Visibility: default

Description: `Window` creates a window on the screen, to which output will be sent. $(X1, Y1)$ are the coordinates of the upper left corner of the window, $(X2, Y2)$ are the coordinates of the bottom right corner of the window. These coordinates are relative to the entire screen, with the top left corner equal to $(1, 1)$. Further coordinate operations, except for the next `Window` call, are relative to the window's top left corner.

Errors: None.

See also: [GotoXY \(591\)](#), [WhereX \(597\)](#), [WhereY \(597\)](#), [ClrScr \(589\)](#)

Listing: `./examples/crtex/ex5.pp`

```

Program Example5;
uses Crt;

{ Program to demonstrate the Window function. }

begin
  ClrScr;
  WriteLn('Creating a window from 30,10 to 50,20');
  Window(30,10,50,20);
  WriteLn('We are now writing in this small window we just created, we '+
    'can"t get outside it when writing long lines like this one');
  Write('Press any key to clear the window');
  ReadKey;
  ClrScr;
  Write('The window is cleared, press any key to restore to fullscreen');
  ReadKey;
  {Full Screen is 80x25}
  Window(1,1,80,25);
  Clrscr;
  WriteLn('Back in Full Screen');
end.

```

40.5 TCharAttr

`TCharAttr = packed record`

```
    ch : AnsiChar;  
    attr : Byte;  
end
```


Chapter 41

Reference for unit 'cthreads'

41.1 Used units

Table 41.1: Used units by unit 'cthreads'

Name	Page
System	1335

41.2 Overview

The `CThreads` unit initializes the system unit's thread management routines with an implementation based on the POSIX thread managing routines in the C library. This assures that C libraries that are thread-aware still work if they are linked to by a FPC program.

It doesn't offer any API by itself: the initialization section of the unit just initializes the `ThreadManager` record in the `System` ([1335](#)) unit. This is done using the `SetCThreadManager` ([601](#)) call

The `cthreads` unit simply needs to be included in the `uses` clause of the program, preferably the very first unit, and the initialization section of the unit will do all the work.

Note that including this unit links your program to the C library of the system.

It makes no sense to use this unit on a non-POSIX system: Windows, OS/2 or DOS, therefor it should always be between an `ifdef` statement:

```
program myprogram;

uses
  {$ifdef unix}cthreads{$endif},
  classes, sysutils;
```

The Lazarus IDE inserts this conditional automatically for each new started program.

41.3 Procedures and functions

41.3.1 SetCThreadManager

Synopsis: Sets the thread manager to the C thread manager.

Declaration: `procedure SetCThreadManager`

Visibility: `default`

Description: `SetCThreadManager` actually sets the thread manager to the C thread manager. It can be called to re-set the thread manager if the thread manager was set to some other thread manager during the life-time of the program.

Chapter 42

Reference for unit 'ctypes'

42.1 Used units

Table 42.1: Used units by unit 'ctypes'

Name	Page
System	1335
unixtype	2229

42.2 Overview

The `ctypes` unit contains the definitions of commonly found C types. It can be used when interfaces to C libraries need to be defined. The types here are correct on all platforms, 32 or 64 bit.

The main advantage of using this file is to make sure that all C header import units use the same definitions for basic C types.

The `h2pas` program can include the `ctypes` unit automatically in the units it generates. The `-C` command-line switch can be used for this.

42.3 Constants, types and variables

42.3.1 Types

```
cbool = UnixType.cbool
```

C boolean (longbool).

```
cchar = UnixType.cchar
```

C character type (No signedness specification, 8 bit integer).

```
cdouble = UnixType.cdouble
```

Double precision floating point type (double).

`cfloat = UnixType.cfloat`

Single precision floating point type (single).

`cint = UnixType.cint`

C integer (commonly 32 bit).

`cint16 = UnixType.cint16`

16-bit signed integer.

`cint32 = UnixType.cint32`

32-bit signed integer (commonly: int).

`cint64 = UnixType.cint64`

64-bit integer.

`cint8 = UnixType.cint8`

8-bit signed integer.

`clong = UnixType.clong`

long integer (32/64 bit, depending on CPU register size).

`clongdouble = Double`

Long precision floating point type (extended/double, depending on CPU).

`clonglong = UnixType.clonglong`

Long (64-bit) integer.

`coff_t = UnixType.TOff`

Generic type to indicate offset.

`cschar = UnixType.cschar`

C signed character type (8 bit signed integer).

`cshort = UnixType.cshort`

Short integer (16 bit).

`csigned = UnixType.csigned`

Signed integer (commonly 32 bit).

`csint = UnixType.csint`

Signed integer (commonly 32 bit).

`csize_t = UnixType.size_t`

Generic type to contain a size of all kinds of structures.

`cslong = UnixType.cslong`

Signed long integer (32/64 bit, depending on CPU register size).

`cslonglong = UnixType.cslonglong`

Signed long (64-bit) integer.

`csshort = UnixType.csshort`

Short signed integer (16 bit).

`cuchar = UnixType.cuchar`

C unsigned character type (8 bit unsigned integer).

`cuint = UnixType.cuint`

Unsigned integer (commonly 32 bit).

`cuint16 = UnixType.cuint16`

16-bit unsigned integer.

`cuint32 = UnixType.cuint32`

32-bit unsigned integer.

`cuint64 = UnixType.cuint64`

Unsigned 64-bit integer.

`cuint8 = UnixType.cuint8`

8-bit unsigned integer.

`culong = UnixType.culong`

Unsigned long integer (32/64 bit, depending on CPU register size).

`culonglong = UnixType.culonglong`

Unsigned long (64-bit) integer.

`cunsigned = UnixType.cunsigned`

Unsigned integer (commonly 32 bit).

`cushort = UnixType.cushort`

Short unsigned integer (16 bit).

`pcbool = UnixType.pcbbool`

Pointer to `cbool` (602) type.

`pcchar = UnixType.pcchar`

Pointer to `cchar` (602) type.

`pcdouble = UnixType.pcdouble`

Pointer to `cdouble` (602) type.

`pcfloat = UnixType.pcfloating`

Pointer to `cfloat` (603) type.

`pcint = UnixType.pcint`

Pointer to `cint` (603) type.

`pcint16 = UnixType.pcint16`

Pointer to `cint16` (603) type.

`pcint32 = UnixType.pcint32`

Pointer to `cint32` (603) type.

`pcint64 = UnixType.pcint64`

Pointer to `cint64` (603) type.

`pcint8 = UnixType.pcint8`

Pointer to `cint8` (603) type.

`pclong = UnixType.pclong`

Pointer to `clong` (603) type.

`pclongdouble = ^clongdouble`

Pointer to `clongdouble` (603) type.

`pclonglong = UnixType.pclonglong`

Pointer to `clonglong` (603) type.

`pcschar = UnixType.pcschar`

Pointer to `cschar` (603) type.

`pcshort = UnixType.pcsshort`

Pointer to `cshort` (603) type.

`pcsigned = UnixType.pcsigned`

Pointer to `csigned` (603) type.

`pcsint = UnixType.pcsint`

Pointer to `csint` (604) type.

`pcsize_t = UnixType.psize_t`

Pointer to generic size type.

`pcslong = UnixType.pcslong`

Pointer to `clong` (604) type.

`pcslonglong = UnixType.pcslonglong`

Pointer to `cslonglong` (604) type.

`pcsshort = UnixType.pcsshort`

Pointer to `csshort` (604) type.

`pcuchar = UnixType.pcuchar`

Pointer to `cuchar` (604) type.

`pcuint = UnixType.pcuint`

Pointer to `cuint` (604) type.

`pcuint16 = UnixType.pcuint16`

Pointer to `cuint16` (604) type.

`pcuint32 = UnixType.pcuint32`

Pointer to `cuint32` (604) type.

`pcuint64 = UnixType.pcuint64`

Pointer to `cuint64` (604) type.

`pcuint8 = UnixType.pcuint8`

Pointer to `cuint8` (604) type.

`pculong = UnixType.pculong`

Pointer to `culong` (604) type.

`pculonglong = UnixType.pculonglong`

Pointer to `culonglong` (604) type.

`pcunsigned = UnixType.punsigned`

Pointer to `cunsigned` (605) type.

`pcushort = UnixType.pcushort`

Pointer to `cushort` (605) type.

Chapter 43

Reference for unit 'cwstring'

43.1 Used units

Table 43.1: Used units by unit 'cwstring'

Name	Page
System	1335

43.2 Overview

The `cstring` unit offers no API by itself: it just initializes the widestring manager record of the system ([1335](#)) unit with an implementation that uses collation and conversion routines which are provided by the C library found on most Unix or Linux systems that are POSIX compliant.

The `cstring` should simply be included in the `uses` clause of the program, preferably as one of the first units, and the initialization section of the unit will do all the work.

Note that including this unit links your program to the C library of the system.

It makes no sense to use this unit on a non-POSIX system like Windows, OS/2 or DOS. Therefore it should always be enclosed with an `ifdef` statement:

```
program myprogram;

uses
  {$ifdef unix}cstring, {$endif}
  classes, sysutils;
```

43.3 Procedures and functions

43.3.1 SetCWidestringManager

Synopsis: Set the Widestring manager of the system unit to the C version.

Declaration: `procedure SetCWidestringManager`

Visibility: `default`

Description: `SetCWidestringManager` actually sets the widestring manager record of the system unit. It is called automatically by the initialization section of the unit.

Chapter 44

Reference for unit 'DateUtils'

44.1 Used units

Table 44.1: Used units by unit 'DateUtils'

Name	Page
Math	1029
System	1335
System.TimeSpan	??
sysutils	1609

44.2 Overview

`DateUtils` contains a large number of date/time manipulation routines, all based on the `TDateTime` type. There are routines for date/time math, for comparing dates and times, for composing dates and decomposing dates in their constituent parts.

44.3 Constants, types and variables

44.3.1 Constants

`ApproxDaysPerMonth : Double = 30.4375`

Average number of days in a month, measured over a year. Used in `MonthsBetween` ([662](#)).

`ApproxDaysPerYear : Double = 365.25`

Average number of days in a year, measured over 4 years. Used in `YearsBetween` ([708](#)).

`DateTimeEpsilon : Double = 1E-10`

`DayFriday = 5`

ISO day number for Friday.

DayMonday = 1

ISO day number for Monday.

DaySaturday = 6

ISO day number for Saturday.

DaysPerWeek = 7

Number of days in a week.

DaysPerYear : Array[Boolean] of Word = (365, 366)

Array with number of days in a year. The Boolean index indicates whether it is a leap year or not.

DaySunday = 7

ISO day number for Sunday.

DayThursday = 4

ISO day number for Thursday.

DayTuesday = 2

ISO day number for Tuesday.

DayWednesday = 3

ISO day number for Wednesday.

MonthApril = 4

ISO month number for April.

MonthAugust = 8

ISO month number for August.

MonthDecember = 12

ISO month number for December.

MonthFebruary = 2

ISO month number for february.

MonthJanuary = 1

ISO month number for january.

`MonthJuly = 7`

ISO month number for July.

`MonthJune = 6`

ISO month number for June.

`MonthMarch = 3`

ISO month number for March.

`MonthMay = 5`

ISO month number for May.

`MonthNovember = 11`

ISO month number for November.

`MonthOctober = 10`

ISO month number for October.

`MonthSeptember = 9`

ISO month number for September.

`MonthsPerYear = 12`

Number of months in a year.

`OneHour = TDateTime(1) / HoursPerDay`

One hour as a fraction of a day (suitable for TDateTime).

`OneMillisecond = TDateTime(1) / MSecsPerDay`

One millisecond as a fraction of a day (suitable for TDateTime).

`OneMinute = TDateTime(1) / MinsPerDay`

One minute as a fraction of a day (suitable for TDateTime).

`OneSecond = TDateTime(1) / SecsPerDay`

One second as a fraction of a day (suitable for TDateTime).

`RecodeLeaveFieldAsIs = High(Word)`

Bitmask deciding what to do with each TDateTime field in recode routines.

`WeeksPerFortnight = 2`

Number of weeks in fortnight.

`YearsPerCentury = 100`

Number of years in a century.

`YearsPerDecade = 10`

Number of years in a decade.

`YearsPerMillennium = 1000`

Number of years in a millennium.

44.3.2 Types

`TLocalTimeType = (lttStandard, lttDaylight, lttAmbiguous, lttInvalid)`

Table 44.2: Enumeration values for type TLocalTimeType

Value	Explanation
<code>lttAmbiguous</code>	
<code>lttDaylight</code>	
<code>lttInvalid</code>	
<code>lttStandard</code>	

44.4 Procedures and functions

44.4.1 CompareDate

Synopsis: Compare 2 dates, disregarding the time of day.

Declaration: `function CompareDate(const A: TDateTime; const B: TDateTime)
: TValueRelationship`

Visibility: default

Description: `CompareDate` compares the date parts of two timestamps A and B and returns the following results:

`< 0` if the day part of A is earlier than the day part of B.

`0` if A and B are the on same day (times may differ) .

`> 0` if the day part of A is later than the day part of B.

See also: [CompareTime \(615\)](#), [CompareDateTime \(614\)](#), [SameDate \(672\)](#), [SameTime \(673\)](#), [SameDateTime \(672\)](#)

Listing: ./examples/datutex/ex99.pp

Program Example99;

```
{ This program demonstrates the CompareDate function }
```

Uses SysUtils,DateUtils;

Const

```
Fmt = 'dddd dd mmmm yyyy';
```

```
Procedure Test(D1,D2 : TDateTime);
```

Var

Cmp : Integer;

begin

```
Write(FormatDateTime(Fmt,D1),' is ');
```

```
Cmp:=CompareDate(D1,D2);
```

If Cmp<0 then

```
write('earlier than ')
```

```
else if Cmp>0 then
```

Write('later than ')

else

```
Write('equal to ');
```

```
WriteIn(FormatDateTime(Fmt,D2));
```

end;

Var

D,N : TDateTime;

Begin

D:=Today;

N:=Now;

```
Test(D,D);
```

Test(N,N);

```
Test(D+1,D);
```

```
Test(D-1,D);
```

```
Test(D+OneSecond,D);
```

```
Test(D-OneSecond,D);
```

```
Test(N+OneSecond,N);
```

```
Test(N-OneSecond,N);
```

End.

44.4.2 CompareDateTime

Synopsis: Compare 2 dates, taking into account the time of day.

```
Declaration: function CompareDateTime(const A: TDateTime; const B: TDateTime)
                                     : TValueRelationship
```

Visibility: default

Description: CompareDateTime compares two timestamps A and B and returns the following results:

Visibility: default

Description: `CompareTime` compares the time parts of two timestamps A and B and returns the following results:

< 0 if the time part of A is earlier than the time part of B.

0 if A and B have the same time part (dates may differ) .

> 0 if the time part of A is later than the time part of B.

See also: `CompareDateTime` (614), `CompareDate` (613), `SameDate` (672), `SameTime` (673), `SameDateTime` (672)

Listing: `./examples/datutex/ex100.pp`

Program Example100;

{ This program demonstrates the CompareTime function }

Uses SysUtils,DateUtils;

Const

 Fmt = 'dddd dd mmmm yyyy hh:nn:ss.zzz';

Procedure Test(D1,D2 : TDateTime);

Var

 Cmp : Integer;

begin

Write(**FormatDateTime**(Fmt,D1),' has ');

 Cmp:=**CompareDateTime**(D1,D2);

If Cmp<0 **then**

write('earlier time than ')

else if Cmp>0 **then**

Write('later time than ')

else

Write('equal time with ');

Writeln(**FormatDateTime**(Fmt,D2));

end;

Var

 D,N : TDateTime;

Begin

 D:=Today;

 N:=**Now**;

 Test(D,D);

 Test(N,N);

 Test(N+1,N);

 Test(N-1,N);

 Test(N+OneSecond,N);

 Test(N-OneSecond,N);

End.

44.4.4 DateInRange

Synopsis: Checks whether a date value is in a given rang.

Declaration: `function DateInRange (ADate: TDate; AStartDate: TDate; AEndDate: TDate;
AInclusive: Boolean) : Boolean`

Visibility: default

Description: `DateInRange` checks whether the value `ADate` lies between `AStartDate` and `AEndDate`, and returns `True` if it is. When `AINclusive` is `True` (the default), then the limits are included. When `AINclusive` is `False`, the limits are excluded. Only the date part of the 3 parameters is considered.

Errors: The `AStartDate` value must be before `AEndDate`, but no check is performed.

See also: `TimeInRange` (683), `DateTimeInRange` (617)

44.4.5 DateOf

Synopsis: Extract the date part from a `TDateTime` indication.

Declaration: `function DateOf (const AValue: TDateTime) : TDateTime`

Visibility: default

Description: `DateOf` extracts the date part from `AValue` and returns the result.

Since the `TDateTime` is actually a double with the date part encoded in the integer part, this operation corresponds to a call to `Int` (610).

See also: `TimeOf` (683), `YearOf` (707), `MonthOf` (661), `DayOf` (620), `HourOf` (635), `MinuteOf` (657), `SecondOf` (675), `MilliSecondOf` (653)

Listing: `./examples/datutex/ex1.pp`

Program Example1;

{ This program demonstrates the DateOf function }

Uses SysUtils, DateUtils;

Begin

WriteIn('Date is: ', **DateTimeToStr**(**DateOf**(**Now**)));

End.

44.4.6 DateTimeInRange

Synopsis: Checks whether a date/time value is in a given range.

Declaration: `function DateTimeInRange (ADateTime: TDateTime;
AStartDateTime: TDateTime;
AEndDateTime: TDateTime; aInclusive: Boolean)
: Boolean`

Visibility: default

Description: `DateTimeInRange` checks whether the value `ADateTime` lies between `AStartDateTime` and `AEndDateTime`, and returns `True` if it is. When `AINclusive` is `True` (the default), then the limits are included. When `AINclusive` is `False`, the limits are excluded.

Errors: The `AStartDateTime` value must be before `AEndDateTime`, but no check is performed.

See also: `DateInRange` (616), `TimeInRange` (683)

44.4.7 DateTimeToDosDateTime

Synopsis: Convert TDateTime format to DOS date/time format.

Declaration: `function DateTimeToDosDateTime(const AValue: TDateTime) : LongInt`

Visibility: default

Description: `DateTimeToDosDatetime` takes `Value`, a `TDateTime` formatted timestamp, and recodes it to a MS-DOS encoded date/time value. This is a longint with the date/time encoded in the bits as:

0-4 Seconds divided by 2

5-10 Minutes

11-15 Hours

16-20 Day

21-24 Month

25-31 Years since 1980

See also: `DosDateTimeToDateTime` ([628](#))

44.4.8 DateTimeToJulianDate

Synopsis: Converts a TDateTime value to a Julian date representation.

Declaration: `function DateTimeToJulianDate(const AValue: TDateTime) : Double`

Visibility: default

Description: `DateTimeToJulianDate` converts the `AValue` date/time indication to a Julian (as opposed to Gregorian) date.

See also: `JulianDateToDateTime` ([652](#)), `TryJulianDateToDateTime` ([693](#)), `DateTimeToModifiedJulianDate` ([619](#)), `TryModifiedJulianDateToDateTime` ([693](#))

44.4.9 DateTimeToMac

Synopsis: Convert a TDateTime timestamp to a Mac timestamp.

Declaration: `function DateTimeToMac(const AValue: TDateTime) : Int64`

Visibility: default

Description: `DateTimeToMac` converts the `TDateTime` value `AValue` to a valid Mac timestamp indication and returns the result.

Errors: None.

See also: `UnixTimeStampToMac` ([694](#)), `MacToDateTime` ([653](#)), `MacTimeStampToUnix` ([652](#))

44.4.10 DateTimeToModifiedJulianDate

Synopsis: Convert a `TDateTime` value to a modified Julian date representation.

Declaration: `function DateTimeToModifiedJulianDate(const AValue: TDateTime) : Double`

Visibility: default

Description: Modified Julian dates are commonly used by astronomers, geodesists, and historians. The date convention is designed to facilitate chronological calculations, numbers all days in consecutive fashion, beginning at a date sufficiently far into the past so as to precede the historical period.

It represents the number of days since November 17, 1858 and is derived by subtracting 2400000.5 days from a Julian Day Number. Julian Day Number is an integer counter of the days beginning at noon 1 January 4713 B.C., or Julian Day Number 0. The Julian Date is simply the extension of the Day Number to include a real fraction of day, allowing a continuous time unit.

MJD modifies this Julian Date in two ways. The MJD begins at midnight rather than noon, in keeping with more standard conventions. Secondly, for simplicity, the first two digits of the Julian Date are removed. This is because, for some three centuries following 17 November 1858, the Julian day lies between 2400000 and 2500000. The MJD drops those first "24" digits. Thus, $MJD = JD - 2400000.5$.

See also: [DateTimeToJulianDate \(618\)](#), [JulianDateToDateTime \(652\)](#), [TryJulianDateToDateTime \(693\)](#), [TryModifiedJulianDateToDateTime \(693\)](#)

44.4.11 DateTimeToUnix

Synopsis: Convert a `TDateTime` value to Unix epoch time.

Declaration: `function DateTimeToUnix(const AValue: TDateTime; AInputIsUTC: Boolean) : Int64`

Visibility: default

Description: `DateTimeToUnix` converts a `TDateTime` value to a epoch time (i.e. the number of seconds elapsed since 1/1/1970).

See also: [UnixToDateTime \(695\)](#)

44.4.12 DateToISO8601

Synopsis: Converts a `TDateTime` value to ISO 8601 date/time format.

Declaration: `function DateToISO8601(const ADate: TDateTime; AInputIsUTC: Boolean) : string`

Visibility: default

Description: `DateToISO8601` is a `String` function used to convert the `TDateTime` value in `ADate` to ISO 8601 date/time notation.

`ADate` contains the native `TDateTime` value converted in the function.

`AInputIsUTC` indicates if the value in `ADate` represents a date/time value for the UTC time zone. When `AInputIsUTC` contains `True`, the 'Z' (Zulu time) time zone designation is used in the converted ISO 8601 value. Otherwise, the time zone is expressed as a positive or negative number of hours and minutes (such as "-04:00") in the return value.

`DateToISO8601` calls `GetLocalTimeOffset` to determine the time zone offset in use on the local computer. The integer offset is used to adjust the value in `ADate` to the UTC time zone when necessary.

The return value contains the adjusted value in `ADate` formatted using the notation:

- `yyyy-mm-ddThh:nn:ss.zzz±hh:nn` or
- `yyyy-mm-ddThh:nn:ss.zzzZ` for Zulu time

Use `ISO8601ToDate` to convert the return value back to a native `TDateTime` type.

See also: `ISO8601ToDate` ([644](#))

44.4.13 DayOf

Synopsis: Extract the day (of month) part from a `TDateTime` value.

Declaration: `function DayOf(const AValue: TDateTime) : Word`

Visibility: default

Description: `DayOf` returns the day of the month part of the `AValue` date/time indication. It is a number between 1 and 31.

For an example, see `YearOf` ([707](#))

See also: `YearOf` ([707](#)), `WeekOf` ([695](#)), `MonthOf` ([661](#)), `HourOf` ([635](#)), `MinuteOf` ([657](#)), `SecondOf` ([675](#)), `MilliSecondOf` ([653](#))

44.4.14 DayOfTheMonth

Synopsis: Extract the day (of month) part of a `TDateTime` value.

Declaration: `function DayOfTheMonth(const AValue: TDateTime) : Word`

Visibility: default

Description: `DayOfTheMonth` returns the number of days that have passed since the start of the month till the moment indicated by `AValue`. This is a one-based number, i.e. the first day of the month will return 1.

For an example, see the `WeekOfTheMonth` ([695](#)) function.

See also: `DayOfTheYear` ([621](#)), `WeekOfTheMonth` ([695](#)), `HourOfTheMonth` ([636](#)), `MinuteOfTheMonth` ([658](#)), `SecondOfTheMonth` ([676](#)), `MilliSecondOfTheMonth` ([654](#))

44.4.15 DayOfTheWeek

Synopsis: Extracts the day of the week from a `TDateTime` value.

Declaration: `function DayOfTheWeek(const AValue: TDateTime) : Word`

Visibility: default

Description: `DayOfTheWeek` returns the number of days that have passed since the start of the week till the moment indicated by `AValue`. This is a one-based number, i.e. the first day of the week will return 1.

See also: [DayOfTheYear \(621\)](#), [DayOfTheMonth \(620\)](#), [HourOfTheWeek \(636\)](#), [MinuteOfTheWeek \(659\)](#), [SecondOfTheWeek \(676\)](#), [MilliSecondOfTheWeek \(655\)](#)

Listing: ./examples/datutex/ex42.pp

Program Example42;

{ This program demonstrates the WeekOfTheMonth function }

Uses SysUtils, DateUtils;

Var

N : TDateTime;

Begin

N:=Now;

Writeln('Day of the Week : ', DayOfTheWeek(N));

Writeln('Hour of the Week : ', HourOfTheWeek(N));

Writeln('Minute of the Week : ', MinuteOfTheWeek(N));

Writeln('Second of the Week : ', SecondOfTheWeek(N));

Writeln('MilliSecond of the Week : ',
MilliSecondOfTheWeek(N));

End.

44.4.16 DayOfTheYear

Synopsis: Extracts the day of the year from a TDateTime value.

Declaration: function DayOfTheYear(const AValue: TDateTime) : Word

Visibility: default

Description: DayOfTheYear returns the number of days that have passed since the start of the year till the moment indicated by AValue. This is a one-based number, i.e. January 1 will return 1.

For an example, see the WeekOfTheYear ([696](#)) function.

See also: [WeekOfTheYear \(696\)](#), [HourOfTheYear \(637\)](#), [MinuteOfTheYear \(659\)](#), [SecondOfTheYear \(677\)](#), [MilliSecondOfTheYear \(655\)](#)

44.4.17 DaysBetween

Synopsis: Number of whole days between two TDateTime values.

Declaration: function DaysBetween(const ANow: TDateTime; const AThen: TDateTime)
: Integer

Visibility: default

Description: DaysBetween returns the number of whole days between ANow and AThen. This means the fractional part of a day (hours, minutes, etc.) is dropped.

See also: [YearsBetween \(708\)](#), [MonthsBetween \(662\)](#), [WeeksBetween \(697\)](#), [HoursBetween \(637\)](#), [MinutesBetween \(659\)](#), [SecondsBetween \(677\)](#), [MillisecondsBetween \(656\)](#)

Listing: ./examples/datutex/ex58.pp

Program Example58;

{ This program demonstrates the DaysBetween function }

Uses SysUtils,DateUtils;

Procedure Test(ANow,AThen : TDateTime);

begin

Write('Number of days between ');

Write(DateTimeToStr(AThen),' and ',DateTimeToStr(ANow));

WriteLn(' : ',DaysBetween(ANow,AThen));

end;

Var

 D1,D2 : TDateTime;

Begin

 D1:=**Now**;

 D2:=Today-23/24;

 Test(D1,D2);

 D2:=Today-1;

 Test(D1,D2);

 D2:=Today-25/24;

 Test(D1,D2);

 D2:=Today-26/24;

 Test(D1,D2);

 D2:=Today-5.4;

 Test(D1,D2);

 D2:=Today-2.5;

 Test(D1,D2);

End.

44.4.18 DaysInAMonth

Synopsis: Number of days in a month of a certain year.

Declaration: `function DaysInAMonth(const AYear: Word; const AMonth: Word) : Word`

Visibility: default

Description: `DaysInMonth` returns the number of days in the month `AMonth` in the year `AYear`. The return value takes leap years into account.

See also: `WeeksInAYear` (697), `WeeksInYear` (698), `DaysInYear` (624), `DaysInAYear` (623), `DaysInMonth` (623)

Listing: ./examples/datutex/ex17.pp

Program Example17;

{ This program demonstrates the DaysInAMonth function }

Uses SysUtils,DateUtils;

Var

 Y,M : Word;

```

Begin
  For Y:=1992 to 2010 do
    For M:=1 to 12 do
      Writeln(LongMonthNames[m], ' ', Y, ' has ', DaysInAMonth(Y,M), ' days. ');
End.

```

44.4.19 DaysInAYear

Synopsis: Number of days in a particular year.

Declaration: `function DaysInAYear(const AYear: Word) : Word`

Visibility: default

Description: `DaysInAYear` returns the number of weeks in the year `AYear`. The return value is either 365 or 366.

See also: `WeeksInAYear` ([697](#)), `WeeksInYear` ([698](#)), `DaysInYear` ([624](#)), `DaysInMonth` ([623](#)), `DaysInAMonth` ([622](#))

Listing: `./examples/datutex/ex15.pp`

Program Example15;

{ This program demonstrates the DaysInAYear function }

Uses SysUtils, DateUtils;

Var

Y : Word;

Begin

For Y:=1992 **to** 2010 **do**

Writeln(Y, ' has ', DaysInAYear(Y), ' days.');

End.

44.4.20 DaysInMonth

Synopsis: Return the number of days in the month in which a date occurs.

Declaration: `function DaysInMonth(const AValue: TDateTime) : Word`

Visibility: default

Description: `DaysInMonth` returns the number of days in the month in which `AValue` falls. The return value takes leap years into account.

See also: `WeeksInAYear` ([697](#)), `WeeksInYear` ([698](#)), `DaysInYear` ([624](#)), `DaysInAYear` ([623](#)), `DaysInAMonth` ([622](#))

Listing: `./examples/datutex/ex16.pp`

Program Example16;

{ This program demonstrates the DaysInMonth function }

Uses SysUtils, DateUtils;

Var

Y, M : Word;

Begin

For Y:=1992 to 2010 do

For M:=1 to 12 do

WriteLn(LongMonthNames[m], ' ', Y, ' has ', DaysInMonth(EncodeDate(Y, M, 1)), ' days.');

End.

44.4.21 DaysInYear

Synopsis: Return the number of days in the year in which a date occurs.

Declaration: `function DaysInYear(const AValue: TDateTime) : Word`

Visibility: default

Description: `daysInYear` returns the number of days in the year part of `AValue`. The return value is either 365 or 366.

See also: [WeeksInAYear \(697\)](#), [WeeksInYear \(698\)](#), [DaysInAYear \(623\)](#), [DaysInMonth \(623\)](#), [DaysInAMonth \(622\)](#)

Listing: ./examples/datutex/ex14.pp

Program Example14;

{ This program demonstrates the DaysInYear function }

Uses SysUtils, DateUtils;

Var

Y : Word;

Begin

For Y:=1992 to 2010 do

WriteLn(Y, ' has ', DaysInYear(EncodeDate(Y, 1, 1)), ' days.');

End.

44.4.22 DaySpan

Synopsis: Calculate the approximate number of days between two `TDateTime` values.

Declaration: `function DaySpan(const ANow: TDateTime; const AThen: TDateTime) : Double`

Visibility: default

Description: `DaySpan` returns the number of Days between `ANow` and `AThen`, including any fractional parts of a Day.

See also: [YearSpan \(709\)](#), [MonthSpan \(663\)](#), [WeekSpan \(699\)](#), [HourSpan \(638\)](#), [MinuteSpan \(660\)](#), [SecondSpan \(678\)](#), [MilliSecondSpan \(656\)](#), [DaysBetween \(621\)](#)

Listing: ./examples/datutex/ex66.pp

Program Example66;

{ This program demonstrates the DaySpan function }

Uses SysUtils, DateUtils;

Procedure Test(ANow, AThen : TDateTime);

begin

Write('Number of days between ');

Write(DateTimeToStr(AThen), ' and ', DateTimeToStr(ANow));

WriteLn(' : ', DaySpan(ANow, AThen));

end;

Var

 D1, D2 : TDateTime;

Begin

 D1 := **Now**;

 D2 := Today - 23/24;

 Test(D1, D2);

 D2 := Today - 1;

 Test(D1, D2);

 D2 := Today - 25/24;

 Test(D1, D2);

 D2 := Today - 26/24;

 Test(D1, D2);

 D2 := Today - 5.4;

 Test(D1, D2);

 D2 := Today - 2.5;

 Test(D1, D2);

End.

44.4.23 DecodeDateDay

Synopsis: Decode a TDateTime value in year and year of day.

Declaration: procedure DecodeDateDay(const AValue: TDateTime; out AYear: Word;
 out ADayOfYear: Word)

Visibility: default

Description: DecodeDateDay decomposes the date indication in AValue and returns the various components in AYear, ADayOfYear.

See also: EncodeDateTime (629), EncodeDateMonthWeek (629), EncodeDateWeek (630), EncodeDateDay (628), DecodeDateTime (626), DecodeDateWeek (627), DecodeDateMonthWeek (626)

Listing: ./examples/datutex/ex83.pp

Program Example83;

{ This program demonstrates the DecodeDateDay function }

Uses SysUtils, DateUtils;

Var

Y,DoY : Word;
TS : TDateTime;

Begin

DecodeDateDay(**Now**,Y,DoY);
TS:=EncodeDateDay(Y,DoY);
WriteLn('Today is : ',**DateToStr**(TS));

End.**44.4.24 DecodeDateMonthWeek**

Synopsis: Decode a TDateTime value in a month, week of month and day of week.

Declaration: `procedure DecodeDateMonthWeek(const AValue: TDateTime; out AYear: Word;
out AMonth: Word; out AWeekOfMonth: Word;
out ADayOfWeek: Word)`

Visibility: default

Description: `DecodeDateMonthWeek` decomposes the date indication in `AValue` and returns the various components in `AYear`, `AMonth`, `AWeekOfMonth` and `ADayOfWeek`.

See also: `EncodeDateTime` (629), `EncodeDateMonthWeek` (629), `EncodeDateWeek` (630), `EncodeDateDay` (628), `DecodeDateTime` (626), `DecodeDateWeek` (627), `DecodeDateDay` (625)

Listing: ./examples/datutex/ex85.pp

Program Example85;

{ This program demonstrates the DecodeDateMonthWeek function }

Uses SysUtils,DateUtils;

Var

Y,M,Wom,Dow : Word;
TS : TDateTime;

Begin

DecodeDateMonthWeek(**Now**,Y,M,WoM,DoW);
TS:=EncodeDateMonthWeek(Y,M,WoM,Dow);
WriteLn('Today is : ',**DateToStr**(TS));

End.**44.4.25 DecodeDateTime**

Synopsis: Decode a TDateTime value in a date and time value.

Declaration: `procedure DecodeDateTime(const AValue: TDateTime; out AYear: Word;
out AMonth: Word; out ADay: Word;
out AHour: Word; out AMinute: Word;
out ASecond: Word; out AMilliSecond: Word)`

Visibility: default

Description: `DecodeDateTime` decomposes the date/time indication in `AValue` and returns the various components in `AYear`, `AMonth`, `ADay`, `AHour`, `AMinute`, `ASecond`, `AMilliSecond`

See also: `EncodeDateTime` (629), `EncodeDateMonthWeek` (629), `EncodeDateWeek` (630), `EncodeDateDay` (628), `DecodeDateWeek` (627), `DecodeDateDay` (625), `DecodeDateMonthWeek` (626)

Listing: `./examples/datutex/ex79.pp`

Program Example79;

{ This program demonstrates the DecodeDateTime function }

Uses SysUtils,DateUtils;

Var

Y,Mo,D,H,Mi,S,MS : Word;
TS : TDateTime;

Begin

DecodeDateTime(**Now**,Y,Mo,D,H,Mi,S,MS);
TS:=EncodeDateTime(Y,Mo,D,H,Mi,S,MS);
WriteLn('Now is : ',**DateTimeToStr**(TS));

End.

44.4.26 DecodeDateWeek

Synopsis: Decode a TDateTime value in a week of year and day of week.

Declaration: `procedure DecodeDateWeek(const AValue: TDateTime; out AYear: Word;
out AWeekOfYear: Word; out ADayOfWeek: Word)`

Visibility: default

Description: `DecodeDateWeek` decomposes the date indication in `AValue` and returns the various components in `AYear`, `AWeekOfYear`, `ADayOfWeek`.

See also: `EncodeDateTime` (629), `EncodeDateMonthWeek` (629), `EncodeDateWeek` (630), `EncodeDateDay` (628), `DecodeDateTime` (626), `DecodeDateDay` (625), `DecodeDateMonthWeek` (626)

Listing: `./examples/datutex/ex81.pp`

Program Example81;

{ This program demonstrates the DecodeDateWeek function }

Uses SysUtils,DateUtils;

Var

Y,W,Dow : Word;
TS : TDateTime;

Begin

DecodeDateWeek(**Now**,Y,W,Dow);
TS:=EncodeDateWeek(Y,W,Dow);
WriteLn('Today is : ',**DateToStr**(TS));

End.

44.4.27 DecodeDayOfWeekInMonth

Synopsis: Decode a TDateTime value in year, month, day of week parts.

Declaration: `procedure DecodeDayOfWeekInMonth(const AValue: TDateTime;
out AYear: Word; out AMonth: Word;
out ANthDayOfWeek: Word;
out ADayOfWeek: Word)`

Visibility: default

Description: `DecodeDayOfWeekInMonth` decodes the date `AValue` in a `AYear`, `AMonth`, `ADayOfWeek` and `ANthDayOfWeek`. (This is the N-th time that this weekday occurs in the month, e.g. the third Saturday of the month.)

See also: `NthDayOfWeek` (663), `EncodeDateMonthWeek` (629), `#rtl.sysutils.DayOfWeek` (1678), `EncodeDayOfWeekInMonth` (630), `TryEncodeDayOfWeekInMonth` (687)

Listing: ./examples/datutex/ex105.pp

Program Example105;

{ This program demonstrates the DecodeDayOfWeekInMonth function }

Uses SysUtils, DateUtils;

Var

Y,M,NDoW,DoW : Word;

D : TDateTime;

Begin

`DecodeDayOfWeekInMonth(Date, Y, M, NDoW, DoW);`

`D:=EncodeDayOfWeekInMonth(Y, M, NDoW, DoW);`

`Write(DateToStr(D), ' is the ', NDoW, '-th ');`

`WriteLn(formatDateTime('dddd', D), ' of the month.');`

End.

44.4.28 DosDateTimeToDateTime

Synopsis: Convert DOS date/time format to TDateTime format.

Declaration: `function DosDateTimeToDateTime(AValue: LongInt) : TDateTime`

Visibility: default

Description: `DosDateTimeToDateTime` takes a DOS encoded date/time `AValue` and recodes it as a `TDateTime` value.

The bit encoding of the DOS date/time is explained in the `DateTimeToDosDateTime` (618) function.

See also: `DateTimeToDosDateTime` (618)

44.4.29 EncodeDateDay

Synopsis: Encodes a year and day of year to a TDateTime value.

Declaration: `function EncodeDateDay(const AYear: Word; const ADayOfYear: Word)
: TDateTime`

Visibility: default

Description: `EncodeDateDay` encodes the values `AYear` and `ADayOfYear` to a date value and returns this value.

For an example, see `DecodeDateDay` (625).

Errors: If any of the arguments is not valid, then an `EConvertError` exception is raised.

See also: `EncodeDateMonthWeek` (629), `DecodeDateDay` (625), `EncodeDateTime` (629), `EncodeDateWeek` (630), `TryEncodeDateTime` (686), `TryEncodeDateMonthWeek` (685), `TryEncodeDateWeek` (687)

44.4.30 EncodeDateMonthWeek

Synopsis: Encodes a year, month, week of month and day of week to a `TDateTime` value.

Declaration:

```
function EncodeDateMonthWeek(const AYear: Word; const AMonth: Word;
                             const AWeekOfMonth: Word;
                             const ADayOfWeek: Word) : TDateTime
```

Visibility: default

Description: `EncodeDateTime` encodes the values `AYearAMonth`, `WeekOfMonth`, `ADayOfWeek`, to a date value and returns this value.

For an example, see `DecodeDateMonthWeek` (626).

Errors: If any of the arguments is not valid, then an `EConvertError` exception is raised.

See also: `DecodeDateMonthWeek` (626), `EncodeDateTime` (629), `EncodeDateWeek` (630), `EncodeDateDay` (628), `TryEncodeDateTime` (686), `TryEncodeDateWeek` (687), `TryEncodeDateMonthWeek` (685), `TryEncodeDateDay` (685), `NthDayOfWeek` (663)

44.4.31 EncodeDateTime

Synopsis: Encodes a `TDateTime` value from all its parts.

Declaration:

```
function EncodeDateTime(const AYear: Word; const AMonth: Word;
                        const ADay: Word; const AHour: Word;
                        const AMinute: Word; const ASecond: Word;
                        const AMilliSecond: Word) : TDateTime
```

Visibility: default

Description: `EncodeDateTime` encodes the values `AYearAMonth`, `ADay`, `AHour`, `AMinute`, `ASecond` and `AMilliSecond` to a date/time value and returns this value.

For an example, see `DecodeDateTime` (626).

Errors: If any of the arguments is not valid, then an `EConvertError` exception is raised.

See also: `DecodeDateTime` (626), `EncodeDateMonthWeek` (629), `EncodeDateWeek` (630), `EncodeDateDay` (628), `TryEncodeDateTime` (686), `TryEncodeDateWeek` (687), `TryEncodeDateDay` (685), `TryEncodeDateMonthWeek` (685)

44.4.32 EncodeDateWeek

Synopsis: Encode a `TDateTime` value from a year, week and day of week triplet.

Declaration:

```
function EncodeDateWeek(const AYear: Word; const AWeekOfYear: Word;
                        const ADayOfWeek: Word) : TDateTime
function EncodeDateWeek(const AYear: Word; const AWeekOfYear: Word)
                        : TDateTime
```

Visibility: default

Description: `EncodeDateWeek` encodes the values `AYear`, `AWeekOfYear` and `ADayOfWeek` to a date value and returns this value.

For an example, see `DecodeDateWeek` (627).

Errors: If any of the arguments is not valid, then an `EConvertError` exception is raised.

See also: `EncodeDateMonthWeek` (629), `DecodeDateWeek` (627), `EncodeDateTime` (629), `EncodeDateDay` (628), `TryEncodeDateTime` (686), `TryEncodeDateWeek` (687), `TryEncodeDateMonthWeek` (685)

44.4.33 EncodeDayOfWeekInMonth

Synopsis: Encodes a year, month, week, day of week specification to a `TDateTime` value.

Declaration:

```
function EncodeDayOfWeekInMonth(const AYear: Word; const AMonth: Word;
                                const ANthDayOfWeek: Word;
                                const ADayOfWeek: Word) : TDateTime
```

Visibility: default

Description: `EncodeDayOfWeekInMonth` encodes `AYear`, `AMonth`, `ADayOfWeek` and `ANthDayOfWeek` to a valid date stamp and returns the result.

`ANthDayOfWeek` is the N-th time that this weekday occurs in the month, e.g. the third Saturday of the month.

For an example, see `DecodeDayOfWeekInMonth` (628).

Errors: If any of the values is not in range, then an `EConvertError` exception will be raised.

See also: `NthDayOfWeek` (663), `EncodeDateMonthWeek` (629), `#rtl.sysutils.DayOfWeek` (1678), `DecodeDayOfWeekInMonth` (628), `TryEncodeDayOfWeekInMonth` (687)

44.4.34 EncodeTimeInterval

Synopsis: Encode an interval as a `TDateTime` value.

Declaration:

```
function EncodeTimeInterval(Hour: Word; Minute: Word; Second: Word;
                            MilliSecond: Word) : TDateTime
```

Visibility: default

Description: `EncodeTimeInterval` encodes a time interval expressed in Hour, Min, Sec, MSec as a `TDateTime` value and returns the value in Time.

Errors: If Min, Sec, MSec do not contain a valid time indication, then an `EConvertError` exception is raised.

See also: `TryEncodeTimeInterval` (688)

44.4.35 EndOfDay

Synopsis: Calculates a TDateTime value representing the end of a specified day.

Declaration: `function EndOfDay(const AYear: Word; const AMonth: Word;
const ADay: Word) : TDateTime; Overload
function EndOfDay(const AYear: Word; const ADayOfYear: Word)
: TDateTime; Overload`

Visibility: default

Description: EndOfDay returns a TDateTime value with the date/time indication of the last moment (23:59:59.999) of the day given by AYear, AMonth, ADay.

The day may also be indicated with a AYear, ADayOfYear pair.

See also: StartOfDay (681), StartOfDay (679), StartOfTheWeek (682), StartOfAWeek (680), StartOfAMonth (679), StartOfTheMonth (682), EndOfTheWeek (634), EndOfAWeek (632), EndOfTheYear (635), EndOfAYear (633), EndOfTheMonth (634), EndOfAMonth (631), EndOfTheDay (633)

Listing: ./examples/datutex/ex39.pp

Program Example39;

{ This program demonstrates the EndOfDay function }

Uses SysUtils, DateUtils;

Const

Fmt = "End of the day : "dd mmmm yyyy hh:nn:ss";

Var

Y,M,D : Word;

Begin

Y:=YearOf(Today);

M:=MonthOf(Today);

D:=DayOf(Today);

WriteLn(FormatDateTime(Fmt, EndOfDay(Y,M,D)));

DecodeDateDay(Today, Y,D);

WriteLn(FormatDateTime(Fmt, EndOfDay(Y,D)));

End.

44.4.36 EndOfAMonth

Synopsis: Calculate a TDateTime value representing the last day of the indicated month.

Declaration: `function EndOfAMonth(const AYear: Word; const AMonth: Word) : TDateTime`

Visibility: default

Description: EndOfAMonth returns a TDateTime value with the date of the last day of the month indicated by the AYear, AMonth pair.

See also: StartOfTheMonth (682), StartOfAMonth (679), EndOfTheMonth (634), EndOfTheYear (635), EndOfAYear (633), StartOfAWeek (680), StartOfTheWeek (682)

Listing: ./examples/datutex/ex31.pp

Program Example31;

{ This program demonstrates the EndOfAMonth function }

Uses SysUtils,DateUtils;

Const

 Fmt = "Last day of this month : "dd mmmm yyyy';

Var

 Y,M : Word;

Begin

 Y:=YearOf(Today);

 M:=MonthOf(Today);

WriteLn(**FormatDateTime**(Fmt,EndOfAMonth(Y,M)));

End.

44.4.37 EndOfAWeek

Synopsis: Return the last moment of day of the week, given a year and a week in the year.

Declaration: `function EndOfAWeek(const AYear: Word; const AWeekOfYear: Word;
 const ADayOfWeek: Word) : TDateTime
function EndOfAWeek(const AYear: Word; const AWeekOfYear: Word)
 : TDateTime`

Visibility: default

Description: EndOfAWeek returns a TDateTime value with the date of the last moment (23:59:59:999) on the indicated day of the week indicated by the AYear, AWeek, ADayOfWeek values.

The default value for ADayOfWeek is 7.

See also: StartOfTheWeek (682), EndOfTheWeek (634), EndOfAWeek (632), StartOfAMonth (679), EndOfTheYear (635), EndOfAYear (633), EndOfTheMonth (634), EndOfAMonth (631)

Listing: ./examples/datutex/ex35.pp

Program Example35;

{ This program demonstrates the EndOfAWeek function }

Uses SysUtils,DateUtils;

Const

 Fmt = "Last day of this week : "dd mmmm yyyy hh:nn:ss';

 Fmt2 = "Last-1 day of this week : "dd mmmm yyyy hh:nn:ss';

Var

 Y,W : Word;

Begin

 Y:=YearOf(Today);

 W:=WeekOf(Today);

WriteLn(**FormatDateTime**(Fmt,EndOfAWeek(Y,W)));

WriteLn(**FormatDateTime**(Fmt2,EndOfAWeek(Y,W,6)));

End.

44.4.38 EndOfYear

Synopsis: Calculate a TDateTime value representing the last day of a year.

Declaration: `function EndOfYear(const AYear: Word) : TDateTime`

Visibility: default

Description: `StartOfYear` returns a TDateTime value with the date of the last day of the year AYear (December 31).

See also: `StartOfYear` (683), `EndOfYear` (635), `EndOfYear` (633), `EndOfTheMonth` (634), `EndOfA-Month` (631), `StartOfAWeek` (680), `StartOfTheWeek` (682)

Listing: ./examples/datutex/ex27.pp

Program Example27;

{ This program demonstrates the EndOfYear function }

Uses SysUtils, DateUtils;

Const

 Fmt = "Last day of this year : "dd mmmm yyyy";

Begin

 Writeln(FormatDateTime(Fmt, EndOfYear(YearOf(Today))));

End.

44.4.39 EndOfDay

Synopsis: Calculate a TDateTime value that represents the end of a given day.

Declaration: `function EndOfDay(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: `EndOfDay` extracts the date part of AValue and returns a TDateTime value with the date/time indication of the last moment (23:59:59.999) of this day.

See also: `StartOfDay` (681), `StartOfDay` (679), `StartOfTheWeek` (682), `StartOfAWeek` (680), `StartOfA-Month` (679), `StartOfTheMonth` (682), `EndOfTheWeek` (634), `EndOfAWeek` (632), `EndOfTheYear` (635), `EndOfYear` (633), `EndOfTheMonth` (634), `EndOfAMonth` (631), `EndOfDay` (631)

Listing: ./examples/datutex/ex37.pp

Program Example37;

{ This program demonstrates the EndOfDay function }

Uses SysUtils, DateUtils;

Const

 Fmt = "End of the day : "dd mmmm yyyy hh:nn:ss";

Begin

 Writeln(FormatDateTime(Fmt, EndOfDay(Today)));

End.

44.4.40 EndOfTheMonth

Synopsis: Calculate a TDateTime value representing the last day of the month, given a day in that month.

Declaration: `function EndOfTheMonth(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: `EndOfTheMonth` extracts the year and month parts of `AValue` and returns a `TDateTime` value with the date of the first day of that year and month as the `EndOfAMonth` (631) function.

See also: `StartOfAMonth` (679), `StartOfTheMonth` (682), `EndOfAMonth` (631), `EndOfTheYear` (635), `EndOfAYear` (633), `StartOfAWeek` (680), `StartOfTheWeek` (682)

Listing: `./examples/datutex/ex29.pp`

Program Example29;

{ This program demonstrates the EndOfTheMonth function }

Uses SysUtils,DateUtils;

Const

 Fmt = "last day of this month : "dd mmmm yyyy';

Begin

WriteLn(**FormatDateTime**(Fmt,EndOfTheMonth(Today)));

End.

44.4.41 EndOfTheWeek

Synopsis: Calculate a TDateTime value which represents the end of a week, given a date in that week.

Declaration: `function EndOfTheWeek(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: `EndOfTheWeek` extracts the year and week parts of `AValue` and returns a `TDateTime` value with the date of the last day of that week as the `EndOfAWeek` (632) function.

See also: `StartOfAWeek` (680), `StartOfTheWeek` (682), `EndOfAWeek` (632), `StartOfAMonth` (679), `EndOfTheYear` (635), `EndOfAYear` (633), `EndOfTheMonth` (634), `EndOfAMonth` (631)

Listing: `./examples/datutex/ex33.pp`

Program Example33;

{ This program demonstrates the EndOfTheWeek function }

Uses SysUtils,DateUtils;

Const

 Fmt = "last day of this week : "dd mmmm yyyy';

Begin

WriteLn(**FormatDateTime**(Fmt,EndOfTheWeek(Today)));

End.

44.4.42 EndOfTheYear

Synopsis: Calculate a TDateTime value representing the last day of a year, given a date in that year.

Declaration: `function EndOfTheYear(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: `EndOfTheYear` extracts the year part of `AValue` and returns a `TDateTime` value with the date of the last day of that year (December 31), as the `EndOfAYear` (633) function.

See also: `StartOfAYear` (681), `StartOfTheYear` (683), `EndOfTheMonth` (634), `EndOfAMonth` (631), `StartOfAWeek` (680), `StartOfTheWeek` (682), `EndOfAYear` (633)

Listing: ./examples/datutex/ex25.pp

Program Example25;

{ This program demonstrates the EndOfTheYear function }

Uses SysUtils,DateUtils;

Const

 Fmt = "Last day of this year : "dd mmmm yyyy";

Begin

WriteLn(**FormatDateTime**(Fmt,EndOfTheYear(Today)));

End.

44.4.43 HourOf

Synopsis: Extract the hour part from a TDateTime value.

Declaration: `function HourOf(const AValue: TDateTime) : Word`

Visibility: default

Description: `HourOf` returns the hour of the day part of the `AValue` date/time indication. It is a number between 0 and 23.

For an example, see `YearOf` (707)

See also: `YearOf` (707), `WeekOf` (695), `MonthOf` (661), `DayOf` (620), `MinuteOf` (657), `SecondOf` (675), `MilliSecondOf` (653)

44.4.44 HourOfTheDay

Synopsis: Calculate the hour of a given TDateTime value.

Declaration: `function HourOfTheDay(const AValue: TDateTime) : Word`

Visibility: default

Description: `HourOfTheDay` returns the number of hours that have passed since the start of the day till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:59:59 will return 0.

See also: `HourOfTheYear` (637), `HourOfTheMonth` (636), `HourOfTheWeek` (636), `MinuteOfTheDay` (657), `SecondOfTheDay` (675), `MilliSecondOfTheDay` (653)

Listing: ./examples/datutex/ex43.pp

Program Example43;

{ This program demonstrates the HourOfTheDay function }

Uses SysUtils,DateUtils;

Var

N : TDateTime;

Begin

N:=Now;

WriteLn('Hour of the Day : ',HourOfTheDay(N));

WriteLn('Minute of the Day : ',MinuteOfTheDay(N));

WriteLn('Second of the Day : ',SecondOfTheDay(N));

WriteLn('MilliSecond of the Day : ',
MilliSecondOfTheDay(N));

End.

44.4.45 HourOfTheMonth

Synopsis: Calculate the number of hours passed since the start of the month.

Declaration: `function HourOfTheMonth(const AValue: TDateTime) : Word`

Visibility: default

Description: `HourOfTheMonth` returns the number of hours that have passed since the start of the month till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:59:59 on the first day of the month will return 0.

For an example, see the `WeekOfTheMonth` (695) function.

See also: `WeekOfTheMonth` (695), `DayOfTheMonth` (620), `MinuteOfTheMonth` (658), `SecondOfTheMonth` (676), `MilliSecondOfTheMonth` (654)

44.4.46 HourOfTheWeek

Synopsis: Calculate the number of hours elapsed since the start of the week.

Declaration: `function HourOfTheWeek(const AValue: TDateTime) : Word`

Visibility: default

Description: `HourOfTheWeek` returns the number of hours that have passed since the start of the Week till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:59:59 on the first day of the week will return 0.

For an example, see the `DayOfTheWeek` (620) function.

See also: `HourOfTheYear` (637), `HourOfTheMonth` (636), `HourOfTheDay` (635), `DayOfTheWeek` (620), `MinuteOfTheWeek` (659), `SecondOfTheWeek` (676), `MilliSecondOfTheWeek` (655)

44.4.47 HourOfTheYear

Synopsis: Calculate the number of hours passed since the start of the year.

Declaration: `function HourOfTheYear(const AValue: TDateTime) : Word`

Visibility: default

Description: `HourOfTheYear` returns the number of hours that have passed since the start of the year (January 1, 00:00:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. January 1 00:59:59 will return 0.

For an example, see the `WeekOfTheYear` (696) function.

See also: `WeekOfTheYear` (696), `DayOfTheYear` (621), `MinuteOfTheYear` (659), `SecondOfTheYear` (677), `MilliSecondOfTheYear` (655)

44.4.48 HoursBetween

Synopsis: Calculate the number of whole hours between two `TDateTime` values.

Declaration: `function HoursBetween(const ANow: TDateTime; const AThen: TDateTime) : Int64`

Visibility: default

Description: `HoursBetween` returns the number of whole hours between `ANow` and `AThen`. This means the fractional part of an hour (minutes,seconds etc.) is dropped.

See also: `YearsBetween` (708), `MonthsBetween` (662), `WeeksBetween` (697), `DaysBetween` (621), `MinutesBetween` (659), `SecondsBetween` (677), `MillisecondsBetween` (656)

Listing: `./examples/datutex/ex59.pp`

Program Example59;

{ This program demonstrates the HoursBetween function }

Uses SysUtils,DateUtils;

Procedure Test(ANow,AThen : TDateTime);

begin

Write('Number of hours between ');
 Write(DateTimeToStr(AThen),' and ',DateTimeToStr(ANow));
 Writeln(' : ',HoursBetween(ANow,AThen));
end;

Var

 D1,D2 : TDateTime;

Begin

 D1:=Now;
 D2:=D1-(59*OneMinute);
 Test(D1,D2);
 D2:=D1-(61*OneMinute);
 Test(D1,D2);
 D2:=D1-(122*OneMinute);
 Test(D1,D2);
 D2:=D1-(306*OneMinute);

```

Test(D1,D2);
D2:=D1-(5.4*OneHour);
Test(D1,D2);
D2:=D1-(2.5*OneHour);
Test(D1,D2);
End.

```

44.4.49 HourSpan

Synopsis: Calculate the approximate number of hours between two TDateTime values.

Declaration: `function HourSpan(const ANow: TDateTime; const AThen: TDateTime)
: Double`

Visibility: default

Description: HourSpan returns the number of Hours between ANow and AThen, including any fractional parts of a Hour.

See also: YearSpan (709), MonthSpan (663), WeekSpan (699), DaySpan (624), MinuteSpan (660), SecondSpan (678), MilliSecondSpan (656), HoursBetween (637)

Listing: ./examples/datutex/ex67.pp

Program Example67;

{ This program demonstrates the HourSpan function }

Uses SysUtils,DateUtils;

Procedure Test(ANow,AThen : TDateTime);

```

begin
Write('Number of hours between ');
Write(DateTimeToStr(AThen),' and ',DateTimeToStr(ANow));
WriteLn(' : ',HourSpan(ANow,AThen));
end;

```

Var

D1,D2 : TDateTime;

Begin

```

D1:=Now;
D2:=D1-(59*OneMinute);
Test(D1,D2);
D2:=D1-(61*OneMinute);
Test(D1,D2);
D2:=D1-(122*OneMinute);
Test(D1,D2);
D2:=D1-(306*OneMinute);
Test(D1,D2);
D2:=D1-(5.4*OneHour);
Test(D1,D2);
D2:=D1-(2.5*OneHour);
Test(D1,D2);

```

End.

44.4.50 IncDay

Synopsis: Increase a TDateTime value with a number of days.

Declaration: `function IncDay(const AValue: TDateTime; const ANumberOfDays: Integer) : TDateTime`
`function IncDay(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: IncDay adds ANumberOfDays days to AValue and returns the resulting date/time. ANumberOfDays can be positive or negative.

See also: IncYear (642), #rtl.sysutils.IncMonth (1737), IncWeek (641), IncHour (639), IncMinute (640), IncSecond (641), IncMilliSecond (640)

Listing: ./examples/datutex/ex74.pp

Program Example74;

{ This program demonstrates the IncDay function }

Uses SysUtils, DateUtils;

Begin

Writeln('One Day from today is ', **DateToStr**(IncDay(Today, 1)));

Writeln('One Day ago from today is ', **DateToStr**(IncDay(Today, -1)));

End.

44.4.51 IncHour

Synopsis: Increase a TDateTime value with a number of hours.

Declaration: `function IncHour(const AValue: TDateTime; const ANumberOfHours: Int64) : TDateTime`
`function IncHour(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: IncHour adds ANumberOfHours hours to AValue and returns the resulting date/time. ANumberOfHours can be positive or negative.

See also: IncYear (642), #rtl.sysutils.IncMonth (1737), IncWeek (641), IncDay (639), IncMinute (640), IncSecond (641), IncMilliSecond (640)

Listing: ./examples/datutex/ex75.pp

Program Example75

;

{ This program demonstrates the IncHour function }

Uses SysUtils, DateUtils;

Begin

Writeln('One Hour from now is ', **DateTimeToStr**(IncHour(**Now**, 1)));

Writeln('One Hour ago from now is ', **DateTimeToStr**(IncHour(**Now**, -1)));

End.

44.4.52 IncMilliSecond

Synopsis: Increase a TDateTime value with a number of milliseconds.

Declaration: `function IncMilliSecond(const AValue: TDateTime;
const ANumberOfMilliseconds: Int64) : TDateTime
function IncMilliSecond(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: IncMilliSecond adds ANumberOfMilliseconds milliseconds to AValue and returns the resulting date/time. ANumberOfMilliseconds can be positive or negative.

See also: IncYear (642), #rtl.sysutils.IncMonth (1737), IncWeek (641), IncDay (639), IncHour (639), IncSecond (641), IncMilliSecond (640)

Listing: ./examples/datutex/ex78.pp

Program Example78;

{ This program demonstrates the IncMilliSecond function }

Uses SysUtils, DateUtils;

Begin

Writeln('One MilliSecond from now is ', **TimeToStr**(IncMilliSecond(**Now**, 1)));

Writeln('One MilliSecond ago from now is ', **TimeToStr**(IncMilliSecond(**Now**, -1)));

End.

44.4.53 IncMinute

Synopsis: Increase a TDateTime value with a number of minutes.

Declaration: `function IncMinute(const AValue: TDateTime;
const ANumberOfMinutes: Int64) : TDateTime
function IncMinute(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: IncMinute adds ANumberOfMinutes minutes to AValue and returns the resulting date/time. ANumberOfMinutes can be positive or negative.

See also: IncYear (642), #rtl.sysutils.IncMonth (1737), IncWeek (641), IncDay (639), IncHour (639), IncSecond (641), IncMilliSecond (640)

Listing: ./examples/datutex/ex76.pp

Program Example76;

{ This program demonstrates the IncMinute function }

Uses SysUtils, DateUtils;

Begin

Writeln('One Minute from now is ', **TimeToStr**(IncMinute(**Time**, 1)));

Writeln('One Minute ago from now is ', **TimeToStr**(IncMinute(**Time**, -1)));

End.

44.4.54 IncSecond

Synopsis: Increase a TDateTime value with a number of seconds.

Declaration: `function IncSecond(const AValue: TDateTime;
const ANumberOfSeconds: Int64) : TDateTime
function IncSecond(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: IncSecond adds ANumberOfSeconds seconds to AValue and returns the resulting date/time. ANumberOfSeconds can be positive or negative.

See also: IncYear (642), #rtl.sysutils.IncMonth (1737), IncWeek (641), IncDay (639), IncHour (639), IncSecond (641), IncMilliSecond (640)

Listing: ./examples/datutex/ex77.pp

Program Example77;

{ This program demonstrates the IncSecond function }

Uses SysUtils, DateUtils;

Begin

WriteLn('One Second from now is ', **TimeToStr**(IncSecond(**Time**, 1)));

WriteLn('One Second ago from now is ', **TimeToStr**(IncSecond(**Time**, -1)));

End.

44.4.55 IncWeek

Synopsis: Increase a TDateTime value with a number of weeks.

Declaration: `function IncWeek(const AValue: TDateTime; const ANumberOfWeeks: Integer)
: TDateTime
function IncWeek(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: IncWeek adds ANumberOfWeeks weeks to AValue and returns the resulting date/time. ANumberOfWeeks can be positive or negative.

See also: IncYear (642), #rtl.sysutils.IncMonth (1737), IncDay (639), IncHour (639), IncMinute (640), IncSecond (641), IncMilliSecond (640)

Listing: ./examples/datutex/ex73.pp

Program Example73;

{ This program demonstrates the IncWeek function }

Uses SysUtils, DateUtils;

Begin

WriteLn('One Week from today is ', **DateToStr**(IncWeek(**Today**, 1)));

WriteLn('One Week ago from today is ', **DateToStr**(IncWeek(**Today**, -1)));

End.

44.4.56 IncYear

Synopsis: Increase a TDateTime value with a number of years.

```
Declaration: function IncYear(const AValue: TDateTime; const ANumberOfYears: Integer)
              : TDateTime
              function IncYear(const AValue: TDateTime) : TDateTime
```

Visibility: default

Description: IncYear adds ANumberOfYears years to AValue and returns the resulting date/time. ANumberOfYears can be positive or negative.

See also: [#rtl.sysutils.IncMonth \(1737\)](#), [IncWeek \(641\)](#), [IncDay \(639\)](#), [IncHour \(639\)](#), [IncMinute \(640\)](#), [IncSecond \(641\)](#), [IncMilliSecond \(640\)](#)

Listing: ./examples/datutex/ex71.pp

Program Example71;

```
{ This program demonstrates the IncYear function }
```

Uses SysUtils,DateUtils;

Begin

```
WriteLn('One year from today is ',DateToStr(IncYear(Today,1)));
```

```
WriteLn('One year ago from today is ',DateToStr(IncYear(Today,-1)));
```

End.

44.4.57 InvalidDateDayError

Synopsis: Raise an `EConvertError` exception when a day is not a valid day of a year.

Declaration: `procedure InvalidDateDayError(const AYear: Word; const ADayOfYear: Word)`

Visibility: default

Description: `InvalidDateDayError` raises an `EConvertError` (1812) exception and formats the error message with an appropriate description made up from the parts `AYear` and `ADayOfYear`.

Normally this function should not be needed, the conversion routines call it when they have received invalid arguments.

See also: [InvalidDateWeekError \(643\)](#), [InvalidDateTimeError \(643\)](#), [InvalidDateMonthWeekError \(642\)](#), [InvalidDayOfWeekInMonthError \(644\)](#)

44.4.58 InvalidDateMonthWeekError

Synopsis: Raise an `EConvertError` exception when a `Year,Month,WeekOfMonth,DayOfWeek` is invalid.

[illegible]

Visibility: default

Description: `InvalidDateMonthWeekError` raises an `EConvertError` (1812) exception and formats the error message with an appropriate description made up from the parts `AYear`, `AMonth`, `AWeekOfMonth` and `ADayOfWeek`.

Normally this function should not be needed, the conversion routines call it when they have received invalid arguments.

See also: `InvalidDateWeekError` (643), `InvalidDateTimeError` (643), `InvalidDateDayError` (642), `InvalidDay-Of-WeekInMonthError` (644)

44.4.59 InvalidDateTimeError

Synopsis: Raise an `EConvertError` about an invalid date-time specification.

Declaration:

```
procedure InvalidDateTimeError(const AYear: Word; const AMonth: Word;
                               const ADay: Word; const AHour: Word;
                               const AMinute: Word; const ASecond: Word;
                               const AMilliSecond: Word;
                               const ABaseDate: TDateTime)
procedure InvalidDateTimeError(const AYear: Word; const AMonth: Word;
                               const ADay: Word; const AHour: Word;
                               const AMinute: Word; const ASecond: Word;
                               const AMilliSecond: Word)
```

Visibility: default

Description: `InvalidDateTimeError` raises an `EConvertError` (1812) exception and formats the error message with an appropriate description made up from the parts `AYear`, `AMonth`, `ADay`, `AHour`, `AMinute`, `ASecond` and `AMilliSecond`.

Normally this function should not be needed, the conversion routines call it when they have received invalid arguments.

See also: `InvalidDateWeekError` (643), `InvalidDateDayError` (642), `InvalidDateMonthWeekError` (642), `InvalidDayOf-WeekInMonthError` (644)

44.4.60 InvalidDateWeekError

Synopsis: Raise an `EConvertError` with an invalid Year, WeekOfyear and DayOfWeek specification.

Declaration:

```
procedure InvalidDateWeekError(const AYear: Word;
                               const AWeekOfYear: Word;
                               const ADayOfWeek: Word)
```

Visibility: default

Description: `InvalidDateWeekError` raises an `EConvertError` (1812) exception and formats the error message with an appropriate description made up from the parts `AYear`, `AWeek`, `ADayOfWeek`

Normally this function should not be needed, the conversion routines call it when they have received invalid arguments.

See also: `InvalidDateTimeError` (643), `InvalidDateDayError` (642), `InvalidDateMonthWeekError` (642), `InvalidDayOf-WeekInMonthError` (644)

44.4.61 InvalidDayOfWeekInMonthError

Synopsis: Raise an `EConvertError` exception when a `Year,Month,NthDayOfWeek,DayOfWeek` is invalid.

Declaration: `procedure InvalidDayOfWeekInMonthError(const AYear: Word;
const AMonth: Word;
const ANthDayOfWeek: Word;
const ADayOfWeek: Word)`

Visibility: default

Description: `InvalidDayOfWeekInMonthError` raises an `EConvertError` (1812) exception and formats the error message with an appropriate description made up from the parts `AYear`, `AMonth`, `ANthDayOfWeek` and `ADayOfWeek`.

Normally this function should not be needed, the conversion routines call it when they have received invalid arguments.

See also: `InvalidDateWeekError` (643), `InvalidDateTimeError` (643), `InvalidDateDayError` (642), `InvalidDate-MonthWeekError` (642)

44.4.62 IsAM

Declaration: `function IsAM(const AValue: TDateTime) : Boolean`

Visibility: default

44.4.63 IsInLeapYear

Synopsis: Determine whether a date is in a leap year.

Declaration: `function IsInLeapYear(const AValue: TDateTime) : Boolean`

Visibility: default

Description: `IsInLeapYear` returns `True` if the year part of `AValue` is leap year, or `False` if not.

See also: `YearOf` (707), `IsPM` (646), `IsToday` (647), `IsSameDay` (646)

Listing: `./examples/datutex/ex3.pp`

Program Example3;

{ This program demonstrates the IsInLeapYear function }

Uses SysUtils, DateUtils;

Begin

WriteLn('Current year is leap year: ', IsInLeapYear(**Date**));

End.

44.4.64 ISO8601ToDate

Synopsis: Converts a date/time value in ISO 8601 notation to a native `TDateTime` type.

Declaration: `function ISO8601ToDate(const DateString: string; ReturnUTC: Boolean)
: TDateTime`

Visibility: default

Description: `ISO8601ToDate` is a function used to convert a string using ISO 8601 date/time notation to a native `TDatetime` type.

`DateString` contains the date/time value expressed using ISO 8601 notation.

`ReturnUTC` indicates if the `TDatetime` value should be adjusted to reflect the UTC (Coordinated Universal Time) time zone. The default value for the argument is `True`.

`ISO8601ToDate` calls the `TryISO8601ToDate` function to perform the conversion, and raises an `EConvertError` exception if `DateString` contains an invalid ISO 8601 date/time value. The return value contains the native `TDatetime` value for the timestamp (adjusted to UTC when requested).

Use `DateToISO8601` to convert the native date/time value back to its representation using ISO 8601 date/time notation.

See also: `TryISO8601ToDate` (688), `DateToISO8601` (619)

44.4.65 ISO8601ToDateDef

Synopsis: Converts an ISO 8601 date/time string to a `TDatetime` type.

Declaration:

```
function ISO8601ToDateDef(const DateString: string; ReturnUTC: Boolean;
                           aDefault: TDateTime) : TDateTime
function ISO8601ToDateDef(const DateString: string;
                           aDefault: TDateTime; ReturnUTC: Boolean)
                           : TDateTime
```

Visibility: default

Description: `ISO8601ToDateDef` is used to convert a date/time string in ISO 8601 format to its representation as a `TDatetime` value. `DateString` contains the ISO 8601-formatted date/time value converted in the routine. It can use one of the supported ISO 8601 date/time formats, and may contain an optional time zone offset. `DateString` uses the format returned from the `DateToISO8601` function. Each of the following represent the date/time for Noon on July 4, 2019:

- 20190714T08:00:00-04:00
- 2019-07-14T08:00:00-04:00
- 20190714T08:00:00-0400
- 2019-07-14T08:00:00-0400
- 20190714T08:00-04:00
- 20190714T0800-04:00
- 2019-07-14T0800-04:00
- 20190714T0800-0400
- 2019-07-14T0800-0400
- 20190714T12:00:00
- 2019-07-14T12:00:00
- 20190714T12:00:00Z
- 2019-07-14T12:00:00Z
- 20190714T12:00
- 2019-07-14T12:00

- 20190714T12:00Z
- 2019-07-14T12:00Z
- 20190714T1200
- 2019-07-14T1200
- 20190714T1200Z
- 2019-07-14T1200Z

Time values in `DateString` specified without a time zone offset are assumed to be in the local time zone.

`ReturnUTC` indicates if the value in `DateString` is adjusted to UTC (Coordinated Universal Time) in the return value.

`aDefault` contains the default `TDateTime` value used as the return value when `DateString` cannot be successfully parsed and converted.

Please note that the overloaded variant which uses `String`, `Boolean`, and `TDateTime` arguments has been deprecated. Use the variant with `String`, `TDateTime`, and `Boolean` arguments (in that order) instead.

`ISO8601ToDateDef` calls the `TryISO8601ToDate` function in its implementation.

See also: `DateToISO8601` (619), `TryISO8601ToDate` (688)

44.4.66 IsPM

Synopsis: Determine whether a time is PM or AM.

Declaration: `function IsPM(const AValue: TDateTime) : Boolean`

Visibility: default

Description: `IsPM` returns `True` if the time part of `AValue` is later than 12:00 (PM, or afternoon).

See also: `YearOf` (707), `IsInLeapYear` (644), `IsToday` (647), `IsSameDay` (646)

Listing: `./examples/datutex/ex4.pp`

Program Example4;

{ This program demonstrates the IsPM function }

Uses SysUtils, DateUtils;

Begin

WriteLn('Current time is PM : ', IsPM(**Now**));

End.

44.4.67 IsSameDay

Synopsis: Check if two date/time indications are the same day.

Declaration: `function IsSameDay(const AValue: TDateTime; const ABasis: TDateTime) : Boolean`

Visibility: default

Description: `IsSameDay` checks whether `AValue` and `ABasis` have the same date part, and returns `True` if they do, `False` if not.

See also: [Today \(684\)](#), [Yesterday \(710\)](#), [Tomorrow \(684\)](#), [IsToday \(647\)](#)

Listing: `./examples/datutex/ex21.pp`

Program Example21;

{ This program demonstrates the IsSameDay function }

Uses SysUtils, DateUtils;

Var

 I : Integer;
 D : TDateTime;

Begin

For I:=1 **to** 3 **do**

begin

 D:=Today+Random(3)-1;

Write(FormatDateTime('dd mmmm yyyy "is today : "',D));

WriteLn(IsSameDay(D,Today));

end;

End.

44.4.68 IsSameMonth

Synopsis: Check if 2 dates are in the same month.

Declaration: `function IsSameMonth(const AValue: TDateTime; const ABasis: TDateTime) : Boolean`

Visibility: default

Description: `IsSameMonth` will return `True` if the two dates `AValue` and `ABasis` occur in the same year and month. (i.e. if their month and year parts match). Otherwise, `False` is returned.

See also: [IsSameDay \(646\)](#), [IsToday \(647\)](#), [SameDate \(672\)](#)

44.4.69 IsToday

Synopsis: Check whether a given date is today.

Declaration: `function IsToday(const AValue: TDateTime) : Boolean`

Visibility: default

Description: `IsToday` returns `True` if `AValue` is today's date, and `False` otherwise.

See also: [Today \(684\)](#), [Yesterday \(710\)](#), [Tomorrow \(684\)](#), [IsSameDay \(646\)](#)

Listing: `./examples/datutex/ex20.pp`

Program Example20;

{ This program demonstrates the IsToday function }

Uses SysUtils,DateUtils;

Begin

```
Writeln('Today : ',IsToday(Today));
Writeln('Tomorrow : ',IsToday(Tomorrow));
Writeln('Yesterday : ',IsToday(Yesterday));
```

End.

44.4.70 IsValidDate

Synopsis: Check whether a set of values is a valid date indication.

Declaration: `function IsValidDate(const AYear: Word; const AMonth: Word;
const ADay: Word) : Boolean`

Visibility: default

Description: IsValidDate returns True when the values AYear, AMonth, ADay form a valid date indication. If one of the values is not valid (e.g. the day is invalid or does not exist in that particular month), False is returned.

AYear must be in the range 1..9999 to be valid.

See also: IsValidTime (652), IsValidDateTime (650), IsValidDateDay (648), IsValidDateWeek (651), IsValidDateMonthWeek (649)

Listing: ./examples/datutex/ex5.pp

Program Example5;

{ This program demonstrates the IsValidDate function }

Uses SysUtils,DateUtils;

Var

Y,M,D : Word;

Begin

For Y:=2000 to 2004 **do**

For M:=1 to 12 **do**

For D:=1 to 31 **do**

If Not IsValidDate(Y,M,D) **then**

Writeln(D,' is not a valid day in ',Y,'/',M);

End.

44.4.71 IsValidDateDay

Synopsis: Check whether a given year/day of year combination is a valid date.

Declaration: `function IsValidDateDay(const AYear: Word; const ADayOfYear: Word)
: Boolean`

Visibility: default

Description: `IsValidDateDay` returns `True` if `AYear` and `ADayOfYear` form a valid date indication, or `False` otherwise.

`AYear` must be in the range 1..9999 to be valid.

The `ADayOfYear` value is checked to see whether it falls within the valid range of dates for `AYear`.

See also: `IsValidDate` (648), `IsValidTime` (652), `IsValidDateTime` (650), `IsValidDateWeek` (651), `IsValidDateMonthWeek` (649)

Listing: `./examples/datutex/ex9.pp`

Program Example9;

{ This program demonstrates the IsValidDateDay function }

Uses SysUtils, DateUtils;

Var
Y : Word;

Begin
 For Y:=1996 to 2004 **do**
 if IsValidDateDay(Y,366) **then**
 Writeln(Y,' is a leap year');
End.

44.4.72 IsValidDateMonthWeek

Synopsis: Check whether a given year/month/week/day of the week combination is a valid day.

Declaration: `function IsValidDateMonthWeek(const AYear: Word; const AMonth: Word; const AWeekOfMonth: Word; const ADayOfWeek: Word) : Boolean`

Visibility: default

Description: `IsValidDateMonthWeek` returns `True` if `AYear`, `AMonth`, `AWeekOfMonth` and `ADayOfWeek` form a valid date indication, or `False` otherwise.

`AYear` must be in the range 1..9999 to be valid.

The `AWeekOfMonth`, `ADayOfWeek` values are checked to see whether the combination falls within the valid range of weeks for the `AYear`, `AMonth` combination.

See also: `IsValidDate` (648), `IsValidTime` (652), `IsValidDateTime` (650), `IsValidDateDay` (648), `IsValidDateWeek` (651)

Listing: `./examples/datutex/ex11.pp`

Program Example11;

{ This program demonstrates the IsValidDateMonthWeek function }

Uses SysUtils, DateUtils;

Var
Y, W, D : Word;
B : Boolean;

```

Begin
  For Y:=2000 to 2004 do
    begin
      B:=True;
      For W:=4 to 6 do
        For D:=1 to 7 do
          If B then
            begin
              B:=IsValidDateMonthWeek(Y,12,W,D);
            end
          If Not B then
            if (D=1) then
              Writeln('December ',Y,' has exactly ',W,' weeks.')
            else
              Writeln('December ',Y,' has ',W,' weeks and ',D-1,' days.');
            end;
          end;
        end;
      end;
    End.

```

44.4.73 IsValidDateTime

Synopsis: Check whether a set of values is a valid date and time indication.

Declaration: `function IsValidDateTime(const AYear: Word; const AMonth: Word; const ADay: Word; const AHour: Word; const AMinute: Word; const ASecond: Word; const AMilliSecond: Word) : Boolean`

Visibility: default

Description: `IsValidTime` returns `True` when the values `AYear`, `AMonth`, `ADay`, `AHour`, `AMinute`, `ASecond` and `AMilliSecond` form a valid date and time indication. If one of the values is not valid (e.g. the seconds are larger than 60), `False` is returned.

`AYear` must be in the range 1..9999 to be valid.

See also: `IsValidDate` (648), `IsValidTime` (652), `IsValidDateDay` (648), `IsValidDateWeek` (651), `IsValidDateMonthWeek` (649)

Listing: `./examples/datutex/ex7.pp`

Program Example7;

{ This program demonstrates the IsValidDateTime function }

Uses SysUtils, DateUtils;

Var

Y, Mo, D : Word;
H, M, S, MS : Word;
I : Integer;

Begin

For I:=1 to 10 do
 begin
 Y:=2000+Random(5);
 Mo:=Random(15);
 D:=Random(40);

```

H:=Random(32);
M:=Random(90);
S:=Random(90);
MS:=Random(1500);
If Not IsValidDateTime(Y,Mo,D,H,M,S,MS) then
  Writeln(Y,'-',Mo,'-',D,' ',H,':',M,':',S,':',MS,' is not a valid date/time. ');
end;
End.

```

44.4.74 IsValidDateWeek

Synopsis: Check whether a given year/week/day of the week combination is a valid day.

Declaration: `function IsValidDateWeek(const AYear: Word; const AWeekOfYear: Word; const ADayOfWeek: Word) : Boolean`

Visibility: default

Description: `IsValidDateWeek` returns `True` if `AYear`, `AWeekOfYear` and `ADayOfWeek` form a valid date indication, or `False` otherwise.

`AYear` must be in the range 1..9999 to be valid.

The `ADayOfWeek`, `ADayOfWeek` values are checked to see whether the combination falls within the valid range of weeks for `AYear`.

See also: `IsValidDate` (648), `IsValidTime` (652), `IsValidDateTime` (650), `IsValidDateDay` (648), `IsValidDateMonthWeek` (649)

Listing: `./examples/datutex/ex10.pp`

Program Example10;

{ This program demonstrates the IsValidDateWeek function }

Uses SysUtils, DateUtils;

Var

Y, W, D : Word;
B : Boolean;

Begin

For Y:=2000 to 2004 do

begin

B:=True;

For W:=51 to 54 do

For D:=1 to 7 do

If B then

begin

B:=IsValidDateWeek(Y, W, D);

If Not B then

if (D=1) then

Writeln(Y, ' has exactly ', W, ' weeks.')

else

Writeln(Y, ' has ', W, ' weeks and ', D-1, ' days.');

end;

end;

End.

44.4.75 IsValidTime

Synopsis: Check whether a set of values is a valid time indication.

Declaration:

```
function IsValidTime(const AHour: Word; const AMinute: Word;
                    const ASecond: Word; const AMilliSecond: Word)
                    : Boolean
```

Visibility: default

Description: Check whether a set of values is a valid time indication.

44.4.76 JulianDateToDateTime

Synopsis: Convert a Julian date representation to a TDateTime value.

Declaration:

```
function JulianDateToDateTime(const AValue: Double) : TDateTime
```

Visibility: default

Description: JulianDateToDateTime converts the Julian AValue date/time indication to a regular TDateTime date/time indication.

See also: DateTimeToJulianDate (618), TryJulianDateToDateTime (693), DateTimeToModifiedJulianDate (619), TryModifiedJulianDateToDateTime (693)

44.4.77 LocalTimeToUniversal

Synopsis: Convert local time to UTC time.

Declaration:

```
function LocalTimeToUniversal(LT: TDateTime) : TDateTime
function LocalTimeToUniversal(LT: TDateTime; TZOffset: Integer)
                    : TDateTime
```

Visibility: default

Description: LocalTimeToUniversal converts a local time indication to a universal time indication: it undoes the TZOffset time zone offset from the UT Universal time (UTC). If no TZOffset is specified, the local time offset as returned by GetLocalTimeOffset (610) is used.

Note that for times in the past or in the future, or for time zones with DST, omitting the TZOffset may lead to wrong results depending on GetLocalTimeOffset being able to compute the correct offset for the UT on the target platform. Currently only Windows Vista and newer return correct offsets for a given date. Older Windows systems or Linux/Unix return always the offset for the current date.

See also: GetLocalTimeOffset (610), UniversalTimeToLocal (694)

44.4.78 MacTimeStampToUnix

Synopsis: Convert a Mac timestamp to a Unix timestamp.

Declaration:

```
function MacTimeStampToUnix(const AValue: Int64) : Int64
```

Visibility: default

Description: MacTimeStampToUnix converts the Mac timestamp indication in AValue to a UNIX timestamp indication (epoch time)

Errors: None.

See also: UnixTimeStampToMac (694), DateTimeToMac (618), MacToDateTime (653)

44.4.79 MacToDateTime

Synopsis: Convert a Mac timestamp to a TDateTime timestamp.

Declaration: `function MacToDateTime(const AValue: Int64) : TDateTime`

Visibility: default

Description: `MacToDateTime` converts the Mac timestamp indication in `AValue` to a valid `TDateTime` indication.

Errors: None.

See also: `UnixTimeStampToMac` (694), `DateTimeToMac` (618), `MacTimeStampToUnix` (652)

44.4.80 MilliSecondOf

Synopsis: Extract the millisecond part from a TDateTime value.

Declaration: `function MilliSecondOf(const AValue: TDateTime) : Word`

Visibility: default

Description: `MilliSecondOf` returns the second of the minute part of the `AValue` date/time indication. It is a number between 0 and 999.

For an example, see `YearOf` (707)

See also: `YearOf` (707), `WeekOf` (695), `MonthOf` (661), `DayOf` (620), `HourOf` (635), `MinuteOf` (657), `MilliSecondOf` (653)

44.4.81 MilliSecondOfTheDay

Synopsis: Calculate the number of milliseconds elapsed since the start of the day.

Declaration: `function MilliSecondOfTheDay(const AValue: TDateTime) : LongWord`

Visibility: default

Description: `MilliSecondOfTheDay` returns the number of milliseconds that have passed since the start of the Day (00:00:00.000) till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:00:00.000 will return 0.

For an example, see the `HourOfTheDay` (635) function.

See also: `MilliSecondOfTheYear` (655), `MilliSecondOfTheMonth` (654), `MilliSecondOfTheWeek` (655), `MilliSecondOfTheHour` (653), `MilliSecondOfTheMinute` (654), `MilliSecondOfTheSecond` (654), `HourOfTheDay` (635), `MinuteOfTheDay` (657), `SecondOfTheDay` (675)

44.4.82 MilliSecondOfTheHour

Synopsis: Calculate the number of milliseconds elapsed since the start of the hour.

Declaration: `function MilliSecondOfTheHour(const AValue: TDateTime) : LongWord`

Visibility: default

Description: `MilliSecondOfTheHour` returns the number of milliseconds that have passed since the start of the Hour (HH:00:00.000) till the moment indicated by `AValue`. This is a zero-based number, i.e. HH:00:00.000 will return 0.

For an example, see the `MinuteOfTheHour` (658) function.

See also: `MilliSecondOfTheYear` (655), `MilliSecondOfTheMonth` (654), `MilliSecondOfTheWeek` (655), `MilliSecondOfTheDay` (653), `MilliSecondOfTheMinute` (654), `MilliSecondOfTheSecond` (654), `MinuteOfTheHour` (658), `SecondOfTheHour` (675)

44.4.83 MilliSecondOfTheMinute

Synopsis: Calculate the number of milliseconds elapsed since the start of the minute.

Declaration: `function MilliSecondOfTheMinute(const AValue: TDateTime) : LongWord`

Visibility: default

Description: `MilliSecondOfTheMinute` returns the number of milliseconds that have passed since the start of the Minute (HH:MM:00.000) till the moment indicated by `AValue`. This is a zero-based number, i.e. HH:MM:00.000 will return 0.

For an example, see the `SecondOfTheMinute` (675) function.

See also: `MilliSecondOfTheYear` (655), `MilliSecondOfTheMonth` (654), `MilliSecondOfTheWeek` (655), `MilliSecondOfTheDay` (653), `MilliSecondOfTheHour` (653), `MilliSecondOfTheMinute` (654), `MilliSecondOfTheSecond` (654), `SecondOfTheMinute` (675)

44.4.84 MilliSecondOfTheMonth

Synopsis: Calculate number of milliseconds elapsed since the start of the month.

Declaration: `function MilliSecondOfTheMonth(const AValue: TDateTime) : LongWord`

Visibility: default

Description: `MilliSecondOfTheMonth` returns the number of milliseconds that have passed since the start of the month (00:00:00.000) till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:00:00.000 on the first of the month will return 0.

For an example, see the `WeekOfTheMonth` (695) function.

See also: `WeekOfTheMonth` (695), `DayOfTheMonth` (620), `HourOfTheMonth` (636), `MinuteOfTheMonth` (658), `SecondOfTheMonth` (676), `MilliSecondOfTheMonth` (654)

44.4.85 MilliSecondOfTheSecond

Synopsis: Calculate the number of milliseconds elapsed since the start of the second.

Declaration: `function MilliSecondOfTheSecond(const AValue: TDateTime) : Word`

Visibility: default

Description: `MilliSecondOfTheSecond` returns the number of milliseconds that have passed since the start of the second (HH:MM:SS.000) till the moment indicated by `AValue`. This is a zero-based number, i.e. HH:MM:SS.000 will return 0.

See also: [MilliSecondOfTheYear \(655\)](#), [MilliSecondOfTheMonth \(654\)](#), [MilliSecondOfTheWeek \(655\)](#), [MilliSecondOfTheDay \(653\)](#), [MilliSecondOfTheHour \(653\)](#), [MilliSecondOfTheMinute \(654\)](#), [SecondOfTheMinute \(675\)](#)

Listing: ./examples/datutex/ex46.pp

Program Example46;

{ This program demonstrates the MilliSecondOfTheSecond function }

Uses SysUtils, DateUtils;

Var

N : TDateTime;

Begin

N:=Now;

WriteLn('MilliSecond of the Second : ',
MilliSecondOfTheSecond(N));

End.

44.4.86 MilliSecondOfTheWeek

Synopsis: Calculate the number of milliseconds elapsed since the start of the week.

Declaration: `function MilliSecondOfTheWeek(const AValue: TDateTime) : LongWord`

Visibility: default

Description: `MilliSecondOfTheWeek` returns the number of milliseconds that have passed since the start of the Week (00:00:00.000) till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:00:00.000 on the first of the Week will return 0.

For an example, see the [DayOfTheWeek \(620\)](#) function.

See also: [MilliSecondOfTheYear \(655\)](#), [MilliSecondOfTheMonth \(654\)](#), [MilliSecondOfTheDay \(653\)](#), [MilliSecondOfTheHour \(653\)](#), [MilliSecondOfTheMinute \(654\)](#), [MilliSecondOfTheSecond \(654\)](#), [DayOfTheWeek \(620\)](#), [HourOfTheWeek \(636\)](#), [MinuteOfTheWeek \(659\)](#), [SecondOfTheWeek \(676\)](#)

44.4.87 MilliSecondOfTheYear

Synopsis: Calculate the number of milliseconds elapsed since the start of the year.

Declaration: `function MilliSecondOfTheYear(const AValue: TDateTime) : Int64`

Visibility: default

Description: `MilliSecondOfTheYear` returns the number of milliseconds that have passed since the start of the year (January 1, 00:00:00.000) till the moment indicated by `AValue`. This is a zero-based number, i.e. January 1 00:00:00.000 will return 0.

For an example, see the [WeekOfTheYear \(696\)](#) function.

See also: [WeekOfTheYear \(696\)](#), [DayOfTheYear \(621\)](#), [HourOfTheYear \(637\)](#), [MinuteOfTheYear \(659\)](#), [SecondOfTheYear \(677\)](#), [MilliSecondOfTheYear \(655\)](#)

44.4.88 MilliSecondsBetween

Synopsis: Calculate the number of whole milliseconds between two TDateTime values.

Declaration: `function MilliSecondsBetween(const ANow: TDateTime;
const AThen: TDateTime) : Int64`

Visibility: default

Description: `MilliSecondsBetween` returns the number of whole milliseconds between `ANow` and `AThen`. This means a fractional part of a millisecond is dropped.

See also: `YearsBetween` (708), `MonthsBetween` (662), `WeeksBetween` (697), `DaysBetween` (621), `HoursBetween` (637), `MinutesBetween` (659), `SecondsBetween` (677)

Listing: `./examples/datutex/ex62.pp`

Program Example62;

{ This program demonstrates the MilliSecondsBetween function }

Uses SysUtils, DateUtils;

Procedure Test(ANow, AThen : TDateTime);

begin

Write('Number of milliseconds between ');
 Write(**TimeToStr**(AThen), ' and ', **TimeToStr**(ANow));
 WriteLn(' : ', MilliSecondsBetween(ANow, AThen));
end;

Var

 D1, D2 : TDateTime;

Begin

 D1 := **Now**;
 D2 := D1 - (0.9 * OneMilliSecond);
 Test(D1, D2);
 D2 := D1 - (1.0 * OneMilliSecond);
 Test(D1, D2);
 D2 := D1 - (1.1 * OneMilliSecond);
 Test(D1, D2);
 D2 := D1 - (2.5 * OneMilliSecond);
 Test(D1, D2);

End.

44.4.89 MilliSecondSpan

Synopsis: Calculate the approximate number of milliseconds between two TDateTime values.

Declaration: `function MilliSecondSpan(const ANow: TDateTime; const AThen: TDateTime)
: Double`

Visibility: default

Description: `MilliSecondSpan` returns the number of milliseconds between `ANow` and `AThen`. Since millisecond is the smallest fraction of a `TDateTime` indication, the returned number will always be an integer value.

See also: [YearSpan \(709\)](#), [MonthSpan \(663\)](#), [WeekSpan \(699\)](#), [DaySpan \(624\)](#), [HourSpan \(638\)](#), [MinuteSpan \(660\)](#), [SecondSpan \(678\)](#), [MillisecondsBetween \(656\)](#)

Listing: ./examples/datutex/ex70.pp

Program Example70;

{ This program demonstrates the MilliSecondSpan function }

Uses SysUtils,DateUtils;

Procedure Test(ANow,AThen : TDateTime);

begin

Write('Number of milliseconds between ');

Write(**TimeToStr**(AThen),' and ',**TimeToStr**(ANow));

WriteLn(' : ',MilliSecondSpan(ANow,AThen));

end;

Var

 D1,D2 : TDateTime;

Begin

 D1:=**Now**;

 D2:=D1-(0.9*OneMilliSecond);

 Test(D1,D2);

 D2:=D1-(1.0*OneMilliSecond);

 Test(D1,D2);

 D2:=D1-(1.1*OneMilliSecond);

 Test(D1,D2);

 D2:=D1-(2.5*OneMilliSecond);

 Test(D1,D2);

End.

44.4.90 MinuteOf

Synopsis: Extract the minute part from a TDateTime value.

Declaration: function MinuteOf(const AValue: TDateTime) : Word

Visibility: default

Description: MinuteOf returns the minute of the hour part of the AValue date/time indication. It is a number between 0 and 59.

For an example, see YearOf ([707](#))

See also: YearOf ([707](#)), WeekOf ([695](#)), MonthOf ([661](#)), DayOf ([620](#)), HourOf ([635](#)), SecondOf ([675](#)), MilliSecondOf ([653](#))

44.4.91 MinuteOfTheDay

Synopsis: Calculate the number of minutes elapsed since the start of the day.

Declaration: function MinuteOfTheDay(const AValue: TDateTime) : Word

Visibility: default

Description: `MinuteOfDay` returns the number of minutes that have passed since the start of the Day (00:00:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:00:59 will return 0.

For an example, see the `HourOfDay` (635) function.

See also: `MinuteOfYear` (659), `MinuteOfMonth` (658), `MinuteOfWeek` (659), `MinuteOfTheHour` (658), `HourOfDay` (635), `SecondOfDay` (675), `MilliSecondOfDay` (653)

44.4.92 MinuteOfTheHour

Synopsis: Calculate the number of minutes elapsed since the start of the hour.

Declaration: `function MinuteOfTheHour(const AValue: TDateTime) : Word`

Visibility: default

Description: `MinuteOfTheHour` returns the number of minutes that have passed since the start of the Hour (HH:00:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. HH:00:59 will return 0.

See also: `MinuteOfYear` (659), `MinuteOfMonth` (658), `MinuteOfWeek` (659), `MinuteOfDay` (657), `SecondOfTheHour` (675), `MilliSecondOfTheHour` (653)

Listing: `./examples/datutex/ex44.pp`

Program Example44;

{ This program demonstrates the MinuteOfTheHour function }

Uses SysUtils, DateUtils;

Var

 N : TDateTime;

Begin

 N := Now;

 WriteLn('Minute of the Hour : ', MinuteOfTheHour(N));

 WriteLn('Second of the Hour : ', SecondOfTheHour(N));

 WriteLn('MilliSecond of the Hour : ',
 MilliSecondOfTheHour(N));

End.

44.4.93 MinuteOfTheMonth

Synopsis: Calculate number of minutes elapsed since the start of the month.

Declaration: `function MinuteOfTheMonth(const AValue: TDateTime) : Word`

Visibility: default

Description: `MinuteOfTheMonth` returns the number of minutes that have passed since the start of the Month (00:00:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:00:59 on the first day of the month will return 0.

For an example, see the `WeekOfTheMonth` (695) function.

See also: `WeekOfTheMonth` (695), `DayOfTheMonth` (620), `HourOfTheMonth` (636), `MinuteOfTheMonth` (658), `SecondOfTheMonth` (676), `MilliSecondOfTheMonth` (654)

44.4.94 MinuteOfTheWeek

Synopsis: Calculate the number of minutes elapsed since the start of the week.

Declaration: `function MinuteOfTheWeek(const AValue: TDateTime) : Word`

Visibility: default

Description: `MinuteOfTheWeek` returns the number of minutes that have passed since the start of the week (00:00:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:00:59 on the first day of the week will return 0.

For an example, see the `DayOfTheWeek` (620) function.

See also: `MinuteOfTheYear` (659), `MinuteOfTheMonth` (658), `MinuteOfTheDay` (657), `MinuteOfTheHour` (658), `DayOfTheWeek` (620), `HourOfTheWeek` (636), `SecondOfTheWeek` (676), `MilliSecondOfTheWeek` (655)

44.4.95 MinuteOfTheYear

Synopsis: Calculate the number of minutes elapsed since the start of the year.

Declaration: `function MinuteOfTheYear(const AValue: TDateTime) : LongWord`

Visibility: default

Description: `MinuteOfTheYear` returns the number of minutes that have passed since the start of the year (January 1, 00:00:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. January 1 00:00:59 will return 0.

For an example, see the `WeekOfTheYear` (696) function.

See also: `WeekOfTheYear` (696), `DayOfTheYear` (621), `HourOfTheYear` (637), `MinuteOfTheYear` (659), `SecondOfTheYear` (677), `MilliSecondOfTheYear` (655)

44.4.96 MinutesBetween

Synopsis: Calculate the number of whole minutes between two `TDateTime` values.

Declaration: `function MinutesBetween(const ANow: TDateTime; const AThen: TDateTime) : Int64`

Visibility: default

Description: `MinutesBetween` returns the number of whole minutes between `ANow` and `AThen`. This means the fractional part of a minute (seconds, milliseconds etc.) is dropped.

See also: `YearsBetween` (708), `MonthsBetween` (662), `WeeksBetween` (697), `DaysBetween` (621), `HoursBetween` (637), `SecondsBetween` (677), `MillisecondsBetween` (656)

Listing: `./examples/datutex/ex60.pp`

Program Example60;

{ This program demonstrates the MinutesBetween function }

Uses SysUtils, DateUtils;

Procedure Test(ANow, AThen : TDateTime);

```

begin
  Write('Number of minutes between ');
  Write(TimeToStr(AThen),' and ',TimeToStr(ANow));
  Writeln(' : ',MinutesBetween(ANow,AThen));
end;

Var
  D1,D2 : TDateTime;

Begin
  D1:=Now;
  D2:=D1-(59*OneSecond);
  Test(D1,D2);
  D2:=D1-(61*OneSecond);
  Test(D1,D2);
  D2:=D1-(122*OneSecond);
  Test(D1,D2);
  D2:=D1-(306*OneSecond);
  Test(D1,D2);
  D2:=D1-(5.4*OneMinute);
  Test(D1,D2);
  D2:=D1-(2.5*OneMinute);
  Test(D1,D2);
End.

```

44.4.97 MinuteSpan

Synopsis: Calculate the approximate number of minutes between two TDateTime values.

Declaration: `function MinuteSpan(const ANow: TDateTime; const AThen: TDateTime) : Double`

Visibility: default

Description: MinuteSpan returns the number of minutes between ANow and AThen, including any fractional parts of a minute.

See also: YearSpan ([709](#)), MonthSpan ([663](#)), WeekSpan ([699](#)), DaySpan ([624](#)), HourSpan ([638](#)), SecondSpan ([678](#)), MilliSecondSpan ([656](#)), MinutesBetween ([659](#))

Listing: ./examples/datutex/ex68.pp

Program Example68;

{ This program demonstrates the MinuteSpan function }

Uses SysUtils,DateUtils;

Procedure Test(ANow,AThen : TDateTime);

```

begin
  Write('Number of minutes between ');
  Write(TimeToStr(AThen),' and ',TimeToStr(ANow));
  Writeln(' : ',MinuteSpan(ANow,AThen));
end;

```

Var

D1,D2 : TDateTime;

Begin

```
D1:=Now;
D2:=D1-(59*OneSecond);
Test(D1,D2);
D2:=D1-(61*OneSecond);
Test(D1,D2);
D2:=D1-(122*OneSecond);
Test(D1,D2);
D2:=D1-(306*OneSecond);
Test(D1,D2);
D2:=D1-(5.4*OneMinute);
Test(D1,D2);
D2:=D1-(2.5*OneMinute);
Test(D1,D2);
```

End.

44.4.98 ModifiedJulianDateToDateTime

Synopsis: Convert a modified Julian date representation to a TDateTime value.

Declaration: `function ModifiedJulianDateToDateTime(const AValue: Double) : TDateTime`

Visibility: default

Description: Convert a modified Julian date representation to a TDateTime value.

See also: [DateTimeToJulianDate \(618\)](#), [JulianDateToDateTime \(652\)](#), [TryJulianDateToDateTime \(693\)](#), [DateTimeToModifiedJulianDate \(619\)](#), [TryModifiedJulianDateToDateTime \(693\)](#)

44.4.99 MonthOf

Synopsis: Extract the month from a given date.

Declaration: `function MonthOf(const AValue: TDateTime) : Word`

Visibility: default

Description: `MonthOf` returns the month part of the `AValue` date/time indication. It is a number between 1 and 12.

For an example, see [YearOf \(707\)](#)

See also: [YearOf \(707\)](#), [DayOf \(620\)](#), [WeekOf \(695\)](#), [HourOf \(635\)](#), [MinuteOf \(657\)](#), [SecondOf \(675\)](#), [MilliSecondOf \(653\)](#)

44.4.100 MonthOfTheYear

Synopsis: Extract the month of a TDateTime indication.

Declaration: `function MonthOfTheYear(const AValue: TDateTime) : Word`

Visibility: default

Description: `MonthOfTheYear` extracts the month part of `AValue` and returns it. It is an alias for `MonthOf` (661), and is provided for completeness only, corresponding to the other `PartOfTheYear` functions.

For an example, see the `WeekOfTheYear` (696) function.

See also: `MonthOf` (661), `WeekOfTheYear` (696), `DayOfTheYear` (621), `HourOfTheYear` (637), `MinuteOfTheYear` (659), `SecondOfTheYear` (677), `MilliSecondOfTheYear` (655)

44.4.101 MonthsBetween

Synopsis: Calculate the number of whole months between two `TDateTime` values.

Declaration: `function MonthsBetween(const ANow: TDateTime; const AThen: TDateTime; AExact: Boolean) : Integer`

Visibility: default

Description: `MonthsBetween` returns the number of whole months between `ANow` and `AThen`. This number is an approximation, based on an average number of days of 30.4375 per month (average over 4 years). This means the fractional part of a month is dropped.

See also: `YearsBetween` (708), `WeeksBetween` (697), `DaysBetween` (621), `HoursBetween` (637), `MinutesBetween` (659), `SecondsBetween` (677), `MillisecondsBetween` (656)

Listing: `./examples/datutex/ex56.pp`

Program Example56;

{ This program demonstrates the MonthsBetween function }

Uses SysUtils, DateUtils;

Procedure Test(ANow, AThen : TDateTime);

begin

Write('Number of months between ');
 Write(**DateToStr**(AThen), ' and ', **DateToStr**(ANow));
 WriteLn(' : ', MonthsBetween(ANow, AThen));
end;

Var

 D1, D2 : TDateTime;

Begin

 D1 := Today;
 D2 := Today - 364;
 Test(D1, D2);
 D2 := Today - 365;
 Test(D1, D2);
 D2 := Today - 366;
 Test(D1, D2);
 D2 := Today - 390;
 Test(D1, D2);
 D2 := Today - 368;
 Test(D1, D2);
 D2 := Today - 1000;
 Test(D1, D2);

End.

44.4.102 MonthSpan

Synopsis: Calculate the approximate number of months between two TDateTime values.

Declaration: `function MonthSpan(const ANow: TDateTime; const AThen: TDateTime)
: Double`

Visibility: default

Description: MonthSpan returns the number of month between ANow and AThen, including any fractional parts of a month. This number is an approximation, based on an average number of days of 30.4375 per month (average over 4 years).

See also: YearSpan (709), WeekSpan (699), DaySpan (624), HourSpan (638), MinuteSpan (660), SecondSpan (678), MilliSecondSpan (656), MonthsBetween (662)

Listing: ./examples/datutex/ex64.pp

Program Example64;

{ This program demonstrates the MonthSpan function }

Uses SysUtils, DateUtils;

Procedure Test(ANow, AThen : TDateTime);

begin

Write('Number of months between ');

Write(DateToStr(AThen), ' and ', DateToStr(ANow));

WriteLn(' : ', MonthSpan(ANow, AThen));

end;

Var

 D1, D2 : TDateTime;

Begin

 D1 := Today;

 D2 := Today - 364;

 Test(D1, D2);

 D2 := Today - 365;

 Test(D1, D2);

 D2 := Today - 366;

 Test(D1, D2);

 D2 := Today - 390;

 Test(D1, D2);

 D2 := Today - 368;

 Test(D1, D2);

 D2 := Today - 1000;

 Test(D1, D2);

End.

44.4.103 NthDayOfWeek

Synopsis: Calculate which occurrence of weekday in the month a given day represents.

Declaration: `function NthDayOfWeek(const AValue: TDateTime) : Word`

Visibility: default

Description: `NthDayOfWeek` returns the occurrence of the weekday of `AValue` in the month. This is the N-th time that this weekday occurs in the month (e.g. the third Saturday of the month).

See also: `EncodeDateMonthWeek` (629), `#rtl.sysutils.DayOfWeek` (1678), `DecodeDayOfWeekInMonth` (628), `EncodeDayOfWeekInMonth` (630), `TryEncodeDayOfWeekInMonth` (687)

Listing: `./examples/datutex/ex104.pp`

Program `Example104`;

{ This program demonstrates the `NthDayOfWeek` function }

Uses `SysUtils, DateUtils`;

Begin

`Write('Today is the ', NthDayOfWeek(Today), '-th ');`

`WriteLn(formatdateTime('dddd', Today), ' of the month.');`

End.

44.4.104 `PeriodBetween`

Synopsis: Return the period (in years, months, days) between two dates.

Declaration: `procedure PeriodBetween(const ANow: TDateTime; const AThen: TDateTime;
out Years: Word; out months: Word;
out days: Word)`

Visibility: `default`

Description: `PeriodBetween` returns the timespan between 2 dates (`ANow` and `AThen`), expressed as a number of years, months and days in the parameters `Years`, `months` and `days`. Only complete years, months and days are reported.

If `ANow` is before `AThen`, their values are reversed so the result is always positive.

See also: `YearsBetween` (708), `MonthsBetween` (662), `WeeksBetween` (697), `DaysBetween` (621)

44.4.105 `PreviousDayOfWeek`

Synopsis: Given a day of the week, return the previous day of the week.

Declaration: `function PreviousDayOfWeek(DayOfWeek: Word) : Word`

Visibility: `default`

Description: `PreviousDayOfWeek` returns the previous day of the week. If the current day is the first day of the week (1) then the last day will be returned (7).

Remark Note that the days of the week are in ISO notation, i.e. 1-based.

See also: `Yesterday` (710)

Listing: `./examples/datutex/ex22.pp`

Program `Example22`;

{ This program demonstrates the `PreviousDayOfWeek` function }

Uses SysUtils,DateUtils;

Var

D : Word;

Begin

For D:=1 to 7 do

Writeln('Previous day of ',D,' is : ',PreviousDayOfWeek(D));

End.

44.4.106 RecodeDate

Synopsis: Replace date part of a TDateTime value with another date.

Declaration: function RecodeDate(const AValue: TDateTime; const AYear: Word;
const AMonth: Word; const ADay: Word) : TDateTime

Visibility: default

Description: RecodeDate replaces the date part of the timestamp AValue with the date specified in AYear, AMonth, ADay. All other parts (the time part) of the date/time stamp are left untouched.

Errors: If one of the AYear, AMonth, ADay values is not within a valid range then an EConvertError exception is raised.

See also: RecodeYear (671), RecodeMonth (669), RecodeDay (666), RecodeHour (667), RecodeMinute (668), RecodeSecond (670), RecodeDate (665), RecodeTime (670), RecodeDateTime (665)

Listing: ./examples/datutex/ex94.pp

Program Example94;

{ This program demonstrates the RecodeDate function }

Uses SysUtils,DateUtils;

Const

Fmt = 'dddd dd mmmm yyyy hh:nn:ss';

Var

S : AnsiString;

Begin

S:=FormatDateTime(Fmt,RecodeDate(Now,2001,1,1));

Writeln('This moment on the first of the millenium : ',S);

End.

44.4.107 RecodeDateTime

Synopsis: Replace selected parts of a TDateTime value with other values.

Declaration: function RecodeDateTime(const AValue: TDateTime; const AYear: Word;
const AMonth: Word; const ADay: Word;
const AHour: Word; const AMinute: Word;
const ASecond: Word; const AMilliSecond: Word)
: TDateTime

Visibility: default

Description: `RecodeDateTime` replaces selected parts of the timestamp `AValue` with the date/time values specified in `AYear`, `AMonth`, `ADay`, `AHour`, `AMinute`, `ASecond` and `AMilliSecond`. If any of these values equals the predefined constant `RecodeLeaveFieldAsIs` (613), then the corresponding part of the date/time stamp is left untouched.

Errors: If one of the values `AYear`, `AMonth`, `ADay`, `AHour`, `AMinute`, `ASecond` or `AMilliSecond` is not within a valid range (`RecodeLeaveFieldAsIs` **excepted**) then an `EConvertError` exception is raised.

See also: [RecodeYear \(671\)](#), [RecodeMonth \(669\)](#), [RecodeDay \(666\)](#), [RecodeHour \(667\)](#), [RecodeMinute \(668\)](#), [RecodeSecond \(670\)](#), [RecodeMilliSecond \(668\)](#), [RecodeDate \(665\)](#), [RecodeTime \(670\)](#), [TryRecodeDateTime \(693\)](#)

Listing: ./examples/datutex/ex96.pp

Program Example96;

{ This program demonstrates the RecodeDateTime function }

Uses SysUtils,DateUtils;

Const

```
Fmt = 'dddd dd mmmm yyyy hh:nn:ss';
```

Var

```
S : AnsiString;  
D : TDateTime ;
```

Begin

```
D:=Now;  
D:=RecodeDateTime(D,2000,2,RecodeLeaveFieldAsIs,0,0,0,0);  
S:=FormatDateTime(Fmt,D);  
Writeln('This moment in february 2000 : ',S);
```

End.

44.4.108 RecodeDay

Synopsis: Replace day part of a `TDateTime` value with another day.

```
Declaration: function RecodeDay(const AValue: TDateTime; const ADay: Word)
                : TDateTime
```

Visibility: default

Description: `RecodeDay` replaces the `Day` part of the timestamp `AValue` with `ADay`. All other parts of the date/time stamp are left untouched.

Errors: If the `ADay` value is not within a valid range (1 till the number of days in the month) then an `EConvertError` exception is raised.

See also: [RecodeYear \(671\)](#), [RecodeMonth \(669\)](#), [RecodeHour \(667\)](#), [RecodeMinute \(668\)](#), [RecodeSecond \(670\)](#), [RecodeMilliSecond \(668\)](#), [RecodeDate \(665\)](#), [RecodeTime \(670\)](#), [RecodeDateTime \(665\)](#)

Listing: ./examples/datutex/ex89.pp

Program Example89;

```
{ This program demonstrates the RecodeDay function }
```

```
Uses SysUtils,DateUtils;
```

Const

```
Fmt = 'dddd dd mmmm yyyy hh:nn:ss';
```

Var

```
S : AnsiString;
```

Begin

```
S:=FormatDateTime(Fmt,RecodeDay(Now,1));
```

```
WriteIn('This moment on the first of the month : ',S);
```

End.

44.4.109 RecodeHour

: Replace hours part of a `TDateTime` value with another hour.

```
:function RecodeHour(const AValue: TDateTime; const AHour: Word)
                : TDateTime
```

: default

: RecodeHour replaces the Hour part of the timestamp AValue with AHour. All other parts of the date/time stamp are left untouched.

: If the `AHour` value is not within a valid range (0..23) then an `EConvertError` exception is raised.

: RecodeYear (671), RecodeMonth (669), RecodeDay (666), RecodeMinute (668), RecodeSecond (670), RecodeMilliSecond (668), RecodeDate (665), RecodeTime (670), RecodeDateTime (665)

Listing: ./examples/datutex/ex90.pp

Program Example90;

```
{ This program demonstrates the RecodeHour function }
```

```
Uses SysUtils,DateUtils;
```

Const

```
Fmt = 'dddd dd mmmm yyyy hh:nn:ss';
```

Var

```
S : AnsiString;
```

Begin

```
S:=FormatDateTime(Fmt,RecodeHour(Now,0));
```

```
WriteIn('Today, in the first hour : ',S);
```

End.

44.4.110 RecodeMilliSecond

Synopsis: Replace milliseconds part of a `TDateTime` value with another millisecond.

Declaration: `function RecodeMilliSecond(const AValue: TDateTime;
const AMilliSecond: Word) : TDateTime`

Visibility: default

Description: `RecodeMilliSecond` replaces the millisecond part of the timestamp `AValue` with `AMilliSecond`. All other parts of the date/time stamp are left untouched.

Errors: If the `AMilliSecond` value is not within a valid range (0..999) then an `EConvertError` exception is raised.

See also: `RecodeYear` (671), `RecodeMonth` (669), `RecodeDay` (666), `RecodeHour` (667), `RecodeMinute` (668), `RecodeSecond` (670), `RecodeDate` (665), `RecodeTime` (670), `RecodeDateTime` (665)

Listing: `./examples/datutex/ex93.pp`

Program Example93;

{ This program demonstrates the RecodeMilliSecond function }

Uses SysUtils,DateUtils;

Const

 Fmt = 'dddd dd mmmm yyyy hh:nn:ss.zzz';

Var

 S : AnsiString;

Begin

 S:=**FormatDateTime**(Fmt,RecodeMilliSecond(**Now**,0));

WriteLn('This moment, milliseconds stripped : ',S);

End.

44.4.111 RecodeMinute

Synopsis: Replace minutes part of a `TDateTime` value with another minute.

Declaration: `function RecodeMinute(const AValue: TDateTime; const AMinute: Word)
: TDateTime`

Visibility: default

Description: `RecodeMinute` replaces the Minute part of the timestamp `AValue` with `AMinute`. All other parts of the date/time stamp are left untouched.

Errors: If the `AMinute` value is not within a valid range (0..59) then an `EConvertError` exception is raised.

See also: `RecodeYear` (671), `RecodeMonth` (669), `RecodeDay` (666), `RecodeHour` (667), `RecodeSecond` (670), `RecodeMilliSecond` (668), `RecodeDate` (665), `RecodeTime` (670), `RecodeDateTime` (665)

Listing: `./examples/datutex/ex91.pp`

```

Program Example91;
{ This program demonstrates the RecodeMinute function }

Uses SysUtils,DateUtils;

Const
    Fmt = 'dddd dd mmmm yyyy hh:nn:ss';

Var
    S : AnsiString;

Begin
    S:=FormatDateTime(Fmt,RecodeMinute(Now,0));
    WriteLn('This moment in the first minute of the hour: ',S);
End.

```

44.4.112 RecodeMonth

Synopsis: Replace month part of a `TDateTime` value with another month.

```
Declaration: function RecodeMonth(const AValue: TDateTime; const AMonth: Word)
                                     : TDateTime
```

Visibility: default

Description: `RecodeMonth` replaces the `Month` part of the timestamp `AValue` with `AMonth`. All other parts of the date/time stamp are left untouched.

Errors: If the `AMonth` value is not within a valid range (1..12) then an `EConvertError` exception is raised.

See also: RecodeYear (671), RecodeDay (666), RecodeHour (667), RecodeMinute (668), RecodeSecond (670), RecodeMilliSecond (668), RecodeDate (665), RecodeTime (670), RecodeDateTime (665)

Listing: ./examples/datutex/ex88.pp

```
Program Example88;  
  
{ This program demonstrates the RecodeMonth function }  
  
Uses SysUtils,DateUtils;  
  
Const  
    Fmt = 'dddd dd mmmm yyyy hh:nn:ss';  
  
Var  
    S : AnsiString;  
  
Begin  
    S:=FormatDateTime(Fmt,RecodeMonth(Now,5));  
    WriteLn('This moment in May : ',S);  
End.
```

44.4.113 RecodeSecond

Synopsis: Replace seconds part of a `TDateTime` value with another second.

Declaration: `function RecodeSecond(const AValue: TDateTime; const ASecond: Word)
: TDateTime`

Visibility: default

Description: `RecodeSecond` replaces the `Second` part of the timestamp `AValue` with `ASecond`. All other parts of the date/time stamp are left untouched.

Errors: If the `ASecond` value is not within a valid range (0..59) then an `EConvertError` exception is raised.

See also: `RecodeYear` (671), `RecodeMonth` (669), `RecodeDay` (666), `RecodeHour` (667), `RecodeMinute` (668), `RecodeMilliSecond` (668), `RecodeDate` (665), `RecodeTime` (670), `RecodeDateTime` (665)

Listing: `./examples/datutex/ex92.pp`

Program Example92;

{ This program demonstrates the RecodeSecond function }

Uses SysUtils, DateUtils;

Const

 Fmt = 'dddd dd mmmm yyyy hh:nn:ss';

Var

 S : AnsiString;

Begin

 S:=**FormatDateTime**(Fmt, **RecodeSecond**(**Now**, 0));

Writeln('This moment, seconds stripped : ', S);

End.

44.4.114 RecodeTime

Synopsis: Replace time part of a `TDateTime` value with another time.

Declaration: `function RecodeTime(const AValue: TDateTime; const AHour: Word;
 const AMinute: Word; const ASecond: Word;
 const AMilliSecond: Word) : TDateTime`

Visibility: default

Description: `RecodeTime` replaces the time part of the timestamp `AValue` with the date specified in `AHour`, `AMinute`, `ASecond` and `AMilliSecond`. All other parts (the date part) of the date/time stamp are left untouched.

Errors: If one of the values `AHour`, `AMinute`, `ASecond` or `AMilliSecond` is not within a valid range then an `EConvertError` exception is raised.

See also: `RecodeYear` (671), `RecodeMonth` (669), `RecodeDay` (666), `RecodeHour` (667), `RecodeMinute` (668), `RecodeSecond` (670), `RecodeMilliSecond` (668), `RecodeDate` (665), `RecodeDateTime` (665)

Listing: `./examples/datutex/ex95.pp`

Program Example95;

{ This program demonstrates the RecodeTime function }

Uses SysUtils,DateUtils;

Const

 Fmt = 'dddd dd mmmm yyyy hh:nn:ss';

Var

 S : AnsiString;

Begin

 S:=**FormatDateTime**(Fmt,RecodeTime(**Now**,8,0,0,0));

Writeln('Today, 8 AM : ',S);

End.

44.4.115 RecodeYear

Synopsis: Replace year part of a TDateTime value with another year.

Declaration: `function RecodeYear(const AValue: TDateTime; const AYear: Word)
 : TDateTime`

Visibility: default

Description: RecodeYear replaces the year part of the timestamp AValue with AYear. All other parts of the date/time stamp are left untouched.

Errors: If the AYear value is not within a valid range (1..9999) then an EConvertError exception is raised.

See also: RecodeMonth (669), RecodeDay (666), RecodeHour (667), RecodeMinute (668), RecodeSecond (670), RecodeMilliSecond (668), RecodeDate (665), RecodeTime (670), RecodeDateTime (665)

Listing: ./examples/datutex/ex87.pp

Program Example87;

{ This program demonstrates the RecodeYear function }

Uses SysUtils,DateUtils;

Const

 Fmt = 'dddd dd mmmm yyyy hh:nn:ss';

Var

 S : AnsiString;

Begin

 S:=**FormatDateTime**(Fmt,RecodeYear(**Now**,1999));

Writeln('This moment in 1999 : ',S);

End.

44.4.116 SameDate

Synopsis: Check whether two `TDateTime` values have the same date part.

Declaration: `function SameDate(const A: TDateTime; const B: TDateTime) : Boolean`

Visibility: default

Description: `SameDate` compares the date parts of two timestamps A and B and returns `True` if they are equal, `False` if they are not.

The function simply checks whether `CompareDate` (613) returns zero.

See also: `CompareDateTime` (614), `CompareDate` (613), `CompareTime` (615), `SameDateTime` (672), `SameTime` (673)

Listing: `./examples/datutex/ex102.pp`

Program Example102;

{ This program demonstrates the SameDate function }

Uses SysUtils, DateUtils;

Const

 Fmt = 'dddd dd mmmm yyyy hh:nn:ss.zzz';

Procedure Test(D1,D2 : TDateTime);

begin

 Write(FormatDateTime(Fmt,D1),' is the same date as ');

 WriteLn(FormatDateTime(Fmt,D2),' : ',SameDate(D1,D2));

end;

Var

 D,N : TDateTime;

Begin

 D:=Today;

 N:=Now;

 Test(D,D);

 Test(N,N);

 Test(N+1,N);

 Test(N-1,N);

 Test(N+OneSecond,N);

 Test(N-OneSecond,N);

End.

44.4.117 SameDateTime

Synopsis: Check whether two `TDateTime` values have the same date and time parts.

Declaration: `function SameDateTime(const A: TDateTime; const B: TDateTime) : Boolean`

Visibility: default

Description: `SameDateTime` compares the date/time parts of two timestamps A and B and returns `True` if they are equal, `False` if they are not.

The function simply checks whether `CompareDateTime` (614) returns zero.

See also: [CompareDateTime \(614\)](#), [CompareDate \(613\)](#), [CompareTime \(615\)](#), [SameDate \(672\)](#), [SameTime \(673\)](#)

Listing: ./examples/datutex/ex101.pp

Program Example101;

{ This program demonstrates the SameDateTime function }

Uses SysUtils, DateUtils;

Const

 Fmt = 'dddd dd mmmm yyyy hh:nn:ss.zzz';

Procedure Test(D1,D2 : TDateTime);

begin

 Write(FormatDateTime(Fmt,D1),' is the same datetime as ');

 Writeln(FormatDateTime(Fmt,D2),' : ',SameDateTime(D1,D2));

end;

Var

 D,N : TDateTime;

Begin

 D:=Today;

 N:=Now;

 Test(D,D);

 Test(N,N);

 Test(N+1,N);

 Test(N-1,N);

 Test(N+OneSecond,N);

 Test(N-OneSecond,N);

End.

44.4.118 SameTime

Synopsis: Check whether two TDateTime values have the same time part.

Declaration: function SameTime(const A: TDateTime; const B: TDateTime) : Boolean

Visibility: default

Description: SameTime compares the time parts of two timestamps A and B and returns True if they are equal, False if they are not.

The function simply checks whether CompareTime (615) returns zero.

See also: [CompareDateTime \(614\)](#), [CompareDate \(613\)](#), [CompareTime \(615\)](#), [SameDateTime \(672\)](#), [SameDate \(672\)](#)

Listing: ./examples/datutex/ex103.pp

Program Example102;

{ This program demonstrates the SameTime function }

Uses SysUtils, DateUtils;

```

Const
  Fmt = 'dddd dd mmmm yyyy hh:nn:ss.zzz';

Procedure Test(D1,D2 : TDateTime);

begin
  Write(FormatDateTime(Fmt,D1),' is the same time as ');
  Writeln(FormatDateTime(Fmt,D2),' : ',SameTime(D1,D2));
end;

Var
  D,N : TDateTime;

Begin
  D:=Today;
  N:=Now;
  Test(D,D);
  Test(N,N);
  Test(N+1,N);
  Test(N-1,N);
  Test(N+OneSecond,N);
  Test(N-OneSecond,N);
End.

```

44.4.119 ScanDateTime

Synopsis: Scans a string for a TDateTime pattern and returns the date/time.

Declaration: `function ScanDateTime(const Pattern: string; const s: string; const fmt: TFormatSettings; startpos: Integer) : TDateTime; Overload`
`function ScanDateTime(const Pattern: string; const s: string; startpos: Integer) : TDateTime; Overload`

Visibility: default

Description: ScanDateTime scans string S for the date/time pattern Pattern, starting at position StartPos (default 1). Optionally, the format settings fmt can be specified.

In effect, this function does the opposite of what FormatDateTime (1723) does. The Pattern variable must contain a valid date/time pattern: note that not all possible formatdatetime patterns can be recognized, e.g., hn cannot be detected properly.

- There is a special placeholder: a '?' will match any character in the input string.
- Note that the / and : character will be replaced by the actual date and time separator.
- Similarly, the 'am/pm' and 'a/p' will be replaced with the actual AM/PM indicators.
- a TAB character (ASCII Code 9) will match any whitespace block

Errors: In case of an error, a EConvertError (1812) exception is raised.

See also: FormatDateTime (1723)

44.4.120 SecondOf

Synopsis: Extract the second part from a TDateTime value.

Declaration: `function SecondOf(const AValue: TDateTime) : Word`

Visibility: default

Description: `SecondOf` returns the second of the minute part of the `AValue` date/time indication. It is a number between 0 and 59.

For an example, see [YearOf \(707\)](#)

See also: [YearOf \(707\)](#), [WeekOf \(695\)](#), [MonthOf \(661\)](#), [DayOf \(620\)](#), [HourOf \(635\)](#), [MinuteOf \(657\)](#), [MilliSecondOf \(653\)](#)

44.4.121 SecondOfDay

Synopsis: Calculate the number of seconds elapsed since the start of the day.

Declaration: `function SecondOfDay(const AValue: TDateTime) : LongWord`

Visibility: default

Description: `SecondOfDay` returns the number of seconds that have passed since the start of the Day (00:00:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:00:00.999 return 0.

For an example, see the [HourOfDay \(635\)](#) function.

See also: [SecondOfTheYear \(677\)](#), [SecondOfTheMonth \(676\)](#), [SecondOfTheWeek \(676\)](#), [SecondOfTheHour \(675\)](#), [SecondOfTheMinute \(675\)](#), [HourOfDay \(635\)](#), [MinuteOfDay \(657\)](#), [MilliSecondOfTheDay \(653\)](#)

44.4.122 SecondOfTheHour

Synopsis: Calculate the number of seconds elapsed since the start of the hour.

Declaration: `function SecondOfTheHour(const AValue: TDateTime) : Word`

Visibility: default

Description: `SecondOfTheHour` returns the number of seconds that have passed since the start of the Hour (HH:00:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. HH:00:00.999 return 0.

For an example, see the [MinuteOfTheHour \(658\)](#) function.

See also: [SecondOfTheYear \(677\)](#), [SecondOfTheMonth \(676\)](#), [SecondOfTheWeek \(676\)](#), [SecondOfDay \(675\)](#), [SecondOfTheMinute \(675\)](#), [MinuteOfTheHour \(658\)](#), [MilliSecondOfTheHour \(653\)](#)

44.4.123 SecondOfTheMinute

Synopsis: Calculate the number of seconds elapsed since the start of the minute.

Declaration: `function SecondOfTheMinute(const AValue: TDateTime) : Word`

Visibility: default

Description: `SecondOfTheMinute` returns the number of seconds that have passed since the start of the minute (HH:MM:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. HH:MM:00.999 return 0.

See also: `SecondOfTheYear` (677), `SecondOfTheMonth` (676), `SecondOfTheWeek` (676), `SecondOfTheDay` (675), `SecondOfTheHour` (675), `MilliSecondOfTheMinute` (654)

Listing: `./examples/datutex/ex45.pp`

Program Example45;

{ This program demonstrates the SecondOfTheMinute function }

Uses SysUtils, DateUtils;

Var

N : TDateTime;

Begin

N:=Now;

WriteLn('Second of the Minute : ', SecondOfTheMinute(N));

WriteLn('MilliSecond of the Minute : ',
MilliSecondOfTheMinute(N));

End.

44.4.124 SecondOfTheMonth

Synopsis: Calculate number of seconds elapsed since the start of the month.

Declaration: `function SecondOfTheMonth(const AValue: TDateTime) : LongWord`

Visibility: default

Description: `SecondOfTheMonth` returns the number of seconds that have passed since the start of the month (00:00:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:00:00.999 on the first day of the month will return 0.

For an example, see the `WeekOfTheMonth` (695) function.

See also: `WeekOfTheMonth` (695), `DayOfTheMonth` (620), `HourOfTheMonth` (636), `MinuteOfTheMonth` (658), `MilliSecondOfTheMonth` (654)

44.4.125 SecondOfTheWeek

Synopsis: Calculate the number of seconds elapsed since the start of the week.

Declaration: `function SecondOfTheWeek(const AValue: TDateTime) : LongWord`

Visibility: default

Description: `SecondOfTheWeek` returns the number of seconds that have passed since the start of the week (00:00:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. 00:00:00.999 on the first day of the week will return 0.

For an example, see the `DayOfTheWeek` (620) function.

See also: `SecondOfTheYear` (677), `SecondOfTheMonth` (676), `SecondOfTheDay` (675), `SecondOfTheHour` (675), `SecondOfTheMinute` (675), `DayOfTheWeek` (620), `HourOfTheWeek` (636), `MinuteOfTheWeek` (659), `MilliSecondOfTheWeek` (655)

44.4.126 SecondOfTheYear

Synopsis: Calculate the number of seconds elapsed since the start of the year.

Declaration: `function SecondOfTheYear(const AValue: TDateTime) : LongWord`

Visibility: default

Description: `SecondOfTheYear` returns the number of seconds that have passed since the start of the year (January 1, 00:00:00) till the moment indicated by `AValue`. This is a zero-based number, i.e. January 1 00:00:00.999 will return 0.

For an example, see the `WeekOfTheYear` (696) function.

See also: `WeekOfTheYear` (696), `DayOfTheYear` (621), `HourOfTheYear` (637), `MinuteOfTheYear` (659), `SecondOfTheYear` (677), `MilliSecondOfTheYear` (655)

44.4.127 SecondsBetween

Synopsis: Calculate the number of whole seconds between two `TDateTime` values.

Declaration: `function SecondsBetween(const ANow: TDateTime; const AThen: TDateTime) : Int64`

Visibility: default

Description: `SecondsBetween` returns the number of whole seconds between `ANow` and `AThen`. This means the fractional part of a second (milliseconds etc.) is dropped.

See also: `YearsBetween` (708), `MonthsBetween` (662), `WeeksBetween` (697), `DaysBetween` (621), `HoursBetween` (637), `MinutesBetween` (659), `MillisecondsBetween` (656)

Listing: `./examples/datutex/ex61.pp`

Program Example61;

{ This program demonstrates the SecondsBetween function }

Uses SysUtils, DateUtils;

Procedure Test(ANow, AThen : TDateTime);

begin

Write('Number of seconds between ');
 Write(**TimeToStr**(AThen), ' and ', **TimeToStr**(ANow));
 WriteLn(' : ', **SecondsBetween**(ANow, AThen));
end;

Var

 D1, D2 : TDateTime;

Begin

 D1 := **Now**;
 D2 := D1 - (999 * OneMilliSecond);
 Test(D1, D2);
 D2 := D1 - (1001 * OneMilliSecond);
 Test(D1, D2);
 D2 := D1 - (2001 * OneMilliSecond);
 Test(D1, D2);
 D2 := D1 - (5001 * OneMilliSecond);

```

Test(D1,D2);
D2:=D1-(5.4*OneSecond);
Test(D1,D2);
D2:=D1-(2.5*OneSecond);
Test(D1,D2);
End.

```

44.4.128 SecondSpan

Synopsis: Calculate the approximate number of seconds between two TDateTime values.

Declaration: `function SecondSpan(const ANow: TDateTime; const AThen: TDateTime) : Double`

Visibility: default

Description: `SecondSpan` returns the number of seconds between `ANow` and `AThen`, including any fractional parts of a second.

See also: `YearSpan` (709), `MonthSpan` (663), `WeekSpan` (699), `DaySpan` (624), `HourSpan` (638), `MinuteSpan` (660), `MilliSecondSpan` (656), `SecondsBetween` (677)

Listing: `./examples/datutex/ex69.pp`

Program Example69;

{ This program demonstrates the SecondSpan function }

Uses SysUtils, DateUtils;

Procedure Test(ANow, AThen : TDateTime);

```

begin
  Write('Number of seconds between ');
  Write(TimeToStr(AThen), ' and ', TimeToStr(ANow));
  Writeln(' : ', SecondSpan(ANow, AThen));
end;

```

Var

D1, D2 : TDateTime;

Begin

```

D1:=Now;
D2:=D1-(999*OneMilliSecond);
Test(D1,D2);
D2:=D1-(1001*OneMilliSecond);
Test(D1,D2);
D2:=D1-(2001*OneMilliSecond);
Test(D1,D2);
D2:=D1-(5001*OneMilliSecond);
Test(D1,D2);
D2:=D1-(5.4*OneSecond);
Test(D1,D2);
D2:=D1-(2.5*OneSecond);
Test(D1,D2);

```

End.

44.4.129 StartOfDay

Synopsis: Returns the start of a day as a `TDateTime` value, given a day indication.

Declaration:

```
function StartOfDay(const AYear: Word; const AMonth: Word;
                    const ADay: Word) : TDateTime; Overload
function StartOfDay(const AYear: Word; const ADayOfYear: Word)
                    : TDateTime; Overload
```

Visibility: default

Description: `StartOfDay` returns a `TDateTime` value with the date/time indication of the start (0:0:0.000) of the day given by `AYear`, `AMonth`, `ADay`.

The day may also be indicated with a `AYear`, `ADayOfYear` pair.

See also: `StartOfDay` ([681](#)), `StartOfTheWeek` ([682](#)), `StartOfAWeek` ([680](#)), `StartOfAMonth` ([679](#)), `StartOfTheMonth` ([682](#)), `EndOfTheWeek` ([634](#)), `EndOfAWeek` ([632](#)), `EndOfTheYear` ([635](#)), `EndOfAYear` ([633](#)), `EndOfTheMonth` ([634](#)), `EndOfAMonth` ([631](#)), `EndOfTheDay` ([633](#)), `EndOfDay` ([631](#))

Listing: ./examples/datutex/ex38.pp

Program Example38;

{ This program demonstrates the StartOfDay function }

Uses SysUtils, DateUtils;

Const

 Fmt = "Start of the day : "dd mmmm yyyy hh:nn:ss";

Var

 Y,M,D : Word;

Begin

 Y:=YearOf(Today);

 M:=MonthOf(Today);

 D:=DayOf(Today);

 WriteLn(FormatDateTime(Fmt, StartOfDay(Y,M,D)));

 DecodeDateDay(Today, Y, D);

 WriteLn(FormatDateTime(Fmt, StartOfDay(Y,D)));

End.

44.4.130 StartOfAMonth

Synopsis: Return first date of month, given a year/month pair.

Declaration:

```
function StartOfAMonth(const AYear: Word; const AMonth: Word)
                    : TDateTime
```

Visibility: default

Description: `StartOfAMonth` returns a `TDateTime` value with the date of the first day of the month indicated by the `AYear`, `AMonth` pair.

See also: `StartOfTheMonth` ([682](#)), `EndOfTheMonth` ([634](#)), `EndOfAMonth` ([631](#)), `EndOfTheYear` ([635](#)), `EndOfAYear` ([633](#)), `StartOfAWeek` ([680](#)), `StartOfTheWeek` ([682](#))

Listing: ./examples/datutex/ex30.pp

Program Example30;

{ This program demonstrates the StartOfAMonth function }

Uses SysUtils,DateUtils;

Const

 Fmt = "First day of this month : "dd mmmm yyyy';

Var

 Y,M : Word;

Begin

 Y:=YearOf(Today);

 M:=MonthOf(Today);

Writeln(**FormatDateTime**(Fmt,StartOfAMonth(Y,M)));

End.

44.4.131 StartOfAWeek

Synopsis: Return a day of the week, given a year, week and day in the week.

Declaration: `function StartOfAWeek(const AYear: Word; const AWeekOfYear: Word;
 const ADayOfWeek: Word) : TDateTime
function StartOfAWeek(const AYear: Word; const AWeekOfYear: Word)
 : TDateTime`

Visibility: default

Description: StartOfAWeek returns a TDateTime value with the date of the indicated day of the week indicated by the AYear, AWeek, ADayOfWeek values.

The default value for ADayOfWeek is 1.

See also: StartOfTheWeek (682), EndOfTheWeek (634), EndOfAWeek (632), StartOfAMonth (679), EndOfTheYear (635), EndOfAYear (633), EndOfTheMonth (634), EndOfAMonth (631)

Listing: ./examples/datutex/ex34.pp

Program Example34;

{ This program demonstrates the StartOfAWeek function }

Uses SysUtils,DateUtils;

Const

 Fmt = "First day of this week : "dd mmmm yyyy hh:nn:ss';

 Fmt2 = "Second day of this week : "dd mmmm yyyy hh:nn:ss';

Var

 Y,W : Word;

Begin

 Y:=YearOf(Today);

 W:=WeekOf(Today);

Writeln(**FormatDateTime**(Fmt,StartOfAWeek(Y,W)));

Writeln(**FormatDateTime**(Fmt2,StartOfAWeek(Y,W,2)));

End.

44.4.132 StartOfAYear

Synopsis: Return the first day of a given year.

Declaration: `function StartOfAYear(const AYear: Word) : TDateTime`

Visibility: default

Description: `StartOfAYear` returns a `TDateTime` value with the date of the first day of the year `AYear` (January 1).

See also: `StartOfTheYear` (683), `EndOfTheYear` (635), `EndOfAYear` (633), `EndOfTheMonth` (634), `EndOfA-Month` (631), `StartOfAWeek` (680), `StartOfTheWeek` (682)

Listing: `./examples/datutex/ex26.pp`

Program Example26;

{ This program demonstrates the StartOfAYear function }

Uses SysUtils, DateUtils;

Const

`Fmt = "First day of this year : "dd mmmm yyyy";`

Begin

`Writeln(FormatDateTime(Fmt, StartOfAYear(YearOf(Today))));`

End.

44.4.133 StartOfTheDay

Synopsis: Calculate the start of the day as a `TDateTime` value, given a moment in the day.

Declaration: `function StartOfTheDay(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: `StartOfTheDay` extracts the date part of `AValue` and returns a `TDateTime` value with the date/time indication of the start (0:0:0.000) of this day.

See also: `StartOfADay` (679), `StartOfTheWeek` (682), `StartOfAWeek` (680), `StartOfAMonth` (679), `StartOfThe-Month` (682), `EndOfTheWeek` (634), `EndOfAWeek` (632), `EndOfTheYear` (635), `EndOfAYear` (633), `EndOfTheMonth` (634), `EndOfAMonth` (631), `EndOftheDay` (633), `EndOfADay` (631)

Listing: `./examples/datutex/ex36.pp`

Program Example36;

{ This program demonstrates the StartOfTheDay function }

Uses SysUtils, DateUtils;

Const

`Fmt = "Start of the day : "dd mmmm yyyy hh:nn:ss';`

Begin

`Writeln(FormatDateTime(Fmt, StartOfTheDay(Today))));`

End.

44.4.134 StartOfTheMonth

Synopsis: Calculate the first day of the month, given a date in that month.

Declaration: `function StartOfTheMonth(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: `StartOfTheMonth` extracts the year and month parts of `AValue` and returns a `TDateTime` value with the date of the first day of that year and month as the `StartOfAMonth` (679) function.

See also: `StartOfAMonth` (679), `EndOfTheYear` (635), `EndOfAYear` (633), `EndOfTheMonth` (634), `EndOfAMonth` (631), `StartOfAWeek` (680), `StartOfTheWeek` (682)

Listing: ./examples/datutex/ex28.pp

Program Example28;

{ This program demonstrates the StartOfTheMonth function }

Uses SysUtils,DateUtils;

Const

 Fmt = "First day of this month : "dd mmmm yyyy';

Begin

WriteLn(**FormatDateTime**(Fmt,StartOfTheMonth(Today)));

End.

44.4.135 StartOfTheWeek

Synopsis: Return the first day of the week, given a date.

Declaration: `function StartOfTheWeek(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: `StartOfTheWeek` extracts the year and week parts of `AValue` and returns a `TDateTime` value with the date of the first day of that week as the `StartOfAWeek` (680) function.

See also: `StartOfAWeek` (680), `EndOfTheWeek` (634), `EndOfAWeek` (632), `StartOfAMonth` (679), `EndOfTheYear` (635), `EndOfAYear` (633), `EndOfTheMonth` (634), `EndOfAMonth` (631)

Listing: ./examples/datutex/ex32.pp

Program Example32;

{ This program demonstrates the StartOfTheWeek function }

Uses SysUtils,DateUtils;

Const

 Fmt = "First day of this week : "dd mmmm yyyy';

Begin

WriteLn(**FormatDateTime**(Fmt,StartOfTheWeek(Today)));

End.

44.4.136 StartOfTheYear

Synopsis: Return the first day of the year, given a date in this year.

Declaration: `function StartOfTheYear(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: `StartOfTheYear` extracts the year part of `AValue` and returns a `TDateTime` value with the date of the first day of that year (January 1), as the `StartOfAYear` (681) function.

See also: `StartOfAYear` (681), `EndOfTheYear` (635), `EndOfAYear` (633)

Listing: `./examples/datutex/ex24.pp`

Program Example24;

{ This program demonstrates the StartOfTheYear function }

Uses SysUtils, DateUtils;

Const

 Fmt = "First day of this year : "dd mmmm yyyy';

Begin

Writeln(**FormatDateTime**(Fmt, StartOfTheYear(Today)));

End.

44.4.137 TimeInRange

Synopsis: Checks whether a time value is in a given range.

Declaration: `function TimeInRange(ATime: TTime; AStartTime: TTime; AEndTime: TTime; AInclusive: Boolean) : Boolean`

Visibility: default

Description: `TimeInRange` checks whether the value `ATime` lies between `ASStartTime` and `AEndTime`, and returns `True` if it is. When `AINclusive` is `True` (the default), then the limits are included. When `AINclusive` is `False`, the limits are excluded. Only the time part of the 3 parameters is considered.

Errors: The `ASStartTime` value must be before `AEndTime`, but no check is performed.

See also: `DateInRange` (616), `DateTimeInRange` (617)

44.4.138 TimeOf

Synopsis: Extract the time part from a `TDateTime` indication.

Declaration: `function TimeOf(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: `TimeOf` extracts the time part from `AValue` and returns the result.

Since the `TDateTime` is actually a double with the time part encoded in the fractional part, this operation corresponds to a call to `Frac`.

See also: [DateOf \(617\)](#), [YearOf \(707\)](#), [MonthOf \(661\)](#), [DayOf \(620\)](#), [HourOf \(635\)](#), [MinuteOf \(657\)](#), [SecondOf \(675\)](#), [MilliSecondOf \(653\)](#)

Listing: ./examples/datutex/ex2.pp

Program Example2;

{ This program demonstrates the TimeOf function }

Uses SysUtils,DateUtils;

Begin

WriteLn('Time is : ',**TimeToStr**(**TimeOf**(**Now**)));

End.

44.4.139 Today

Synopsis: Return the current date.

Declaration: `function Today : TDateTime`

Visibility: default

Description: `Today` is an alias for the `Date (1673)` function in the `sysutils (1609)` unit.

For an example, see `Yesterday (710)`

See also: `Date (1673)`, `Yesterday (710)`, `Tomorrow (684)`

44.4.140 Tomorrow

Synopsis: Return the next day after today.

Declaration: `function Tomorrow : TDateTime`

Visibility: default

Description: `Tomorrow` returns tomorrow's date. `Tomorrow` is determined from the system clock, i.e. it is `Today (684)` +1.

See also: `Today (684)`, `Yesterday (710)`

Listing: ./examples/datutex/ex19.pp

Program Example19;

{ This program demonstrates the Tomorrow function }

Uses SysUtils,DateUtils;

Begin

WriteLn(**FormatDateTime**('dd mmmm yyyy',**Today**));

WriteLn(**FormatDateTime**('dd mmmm yyyy',**Tomorrow**));

End.

44.4.141 TryEncodeDateDay

Synopsis: Encode a year and day of year to a TDateTime value.

Declaration: `function TryEncodeDateDay(const AYear: Word; const ADayOfYear: Word;
out AValue: TDateTime) : Boolean`

Visibility: default

Description: TryEncodeDateDay encodes the values AYear and ADayOfYear to a date value and returns this value in AValue.

If the encoding was successful, True is returned. False is returned if any of the arguments is not valid.

See also: EncodeDateDay (628), EncodeDateTime (629), EncodeDateMonthWeek (629), EncodeDateWeek (630), TryEncodeDateTime (686), TryEncodeDateMonthWeek (685), TryEncodeDateWeek (687)

Listing: ./examples/datutex/ex84.pp

Program Example84;

{ This program demonstrates the TryEncodeDateDay function }

Uses SysUtils,DateUtils;

Var

Y,DoY : Word;

TS : TDateTime;

Begin

DecodeDateDay(**Now**,Y,DoY);

If TryEncodeDateDay(Y,DoY,TS) **then**

WriteLn('Today is : ',DateToStr(TS))

else

WriteLn('Wrong year/day of year indication');

End.

44.4.142 TryEncodeDateMonthWeek

Synopsis: Encode a year, month, week of month and day of week to a TDateTime value.

Declaration: `function TryEncodeDateMonthWeek(const AYear: Word; const AMonth: Word;
const AWeekOfMonth: Word;
const ADayOfWeek: Word;
out AValue: TDateTime) : Boolean`

Visibility: default

Description: TryEncodeDateMonthWeek encodes the values AYearAMonth, WeekOfMonth,ADayOfWeek, to a date value and returns this value in AValue.

If the encoding was successful, True is returned, False if any of the arguments is not valid.

See also: DecodeDateMonthWeek (626), EncodeDateTime (629), EncodeDateWeek (630), EncodeDateDay (628), EncodeDateMonthWeek (629), TryEncodeDateTime (686), TryEncodeDateWeek (687), TryEncodeDateDay (685), NthDayOfWeek (663)

Listing: ./examples/datutex/ex86.pp

Program Example86;

{ This program demonstrates the TryEncodeDateMonthWeek function }

Uses SysUtils,DateUtils;

Var

Y,M,Wom,Dow : Word;
TS : TDateTime;

Begin

DecodeDateMonthWeek(**Now**,Y,M,WoM,DoW);
If TryEncodeDateMonthWeek(Y,M,WoM,Dow,TS) **then**
 WriteLn('Today is : ',**DateToStr**(TS))
else
 WriteLn('Invalid year/month/week/dow indication');
End.

44.4.143 TryEncodeDateTime

Synopsis: Encode a Year, Month, Day, Hour, minute, seconds, milliseconds tuple to a TDateTime value.

Declaration: `function TryEncodeDateTime(const AYear: Word; const AMonth: Word;
 const ADay: Word; const AHour: Word;
 const AMinute: Word; const ASecond: Word;
 const AMilliSecond: Word;
 out AValue: TDateTime) : Boolean`

Visibility: default

Description: EncodeDateTime encodes the values AYearAMonth, ADay,AHour, AMinute,ASecond and AMilliSecond to a date/time value and returns this value in AValue.

If the date was encoded successfully, True is returned, False is returned if one of the arguments is not valid.

See also: EncodeDateTime (629), EncodeDateMonthWeek (629), EncodeDateWeek (630), EncodeDateDay (628), TryEncodeDateDay (685), TryEncodeDateWeek (687), TryEncodeDateMonthWeek (685)

Listing: ./examples/datutex/ex80.pp

Program Example79;

{ This program demonstrates the TryEncodeDateTime function }

Uses SysUtils,DateUtils;

Var

Y,Mo,D,H,Mi,S,MS : Word;
TS : TDateTime;

Begin

DecodeDateTime(**Now**,Y,Mo,D,H,Mi,S,MS);
If TryEncodeDateTime(Y,Mo,D,H,Mi,S,MS,TS) **then**
 WriteLn('Now is : ',**DateTimeToStr**(TS))
else
 WriteLn('Wrong date/time indication');

End.

44.4.144 TryEncodeDateWeek

Synopsis: Encode a year, week and day of week triplet to a TDateTime value.

Declaration:

```
function TryEncodeDateWeek(const AYear: Word; const AWeekOfYear: Word;
                           out AValue: TDateTime; const ADayOfWeek: Word)
                           : Boolean
function TryEncodeDateWeek(const AYear: Word; const AWeekOfYear: Word;
                           out AValue: TDateTime) : Boolean
```

Visibility: default

Description: TryEncodeDateWeek encodes the values AYear, AWeekOfYear and ADayOfWeek to a date value and returns this value in AValue.

If the encoding was successful, True is returned. False is returned if any of the arguments is not valid.

See also: EncodeDateMonthWeek (629), EncodeDateWeek (630), EncodeDateTime (629), EncodeDateDay (628), TryEncodeDateTime (686), TryEncodeDateMonthWeek (685), TryEncodeDateDay (685)

Listing: ./examples/datutex/ex82.pp

Program Example82;

{ This program demonstrates the TryEncodeDateWeek function }

Uses SysUtils, DateUtils;

Var

Y, W, Dow : Word;
TS : TDateTime;

Begin

DecodeDateWeek(**Now**, Y, W, Dow);
If TryEncodeDateWeek(Y, W, TS, Dow) **then**
 WriteLn('Today is : ', **DateToStr**(TS))
else
 WriteLn('Invalid date/week indication');

End.

44.4.145 TryEncodeDayOfWeekInMonth

Synopsis: Encode a year, month, week, day of week triplet to a TDateTime value.

Declaration:

```
function TryEncodeDayOfWeekInMonth(const AYear: Word;
                                   const AMonth: Word;
                                   const ANthDayOfWeek: Word;
                                   const ADayOfWeek: Word;
                                   out AValue: TDateTime) : Boolean
```

Visibility: default

Description: `EncodeDayOfWeekInMonth` encodes `AYear`, `AMonth`, `ADayOfWeek` and `ANthDayOfWeek` to a valid date stamp and returns the result in `AValue`.

`ANthDayOfWeek` is the N-th time that this weekday occurs in the month, e.g. the third Saturday of the month.

The function returns `True` if the encoding was successful, `False` if any of the values is not in range.

See also: `NthDayOfWeek` (663), `EncodeDateMonthWeek` (629), `#rtl.sysutils.DayOfWeek` (1678), `DecodeDayOfWeekInMonth` (628), `EncodeDayOfWeekInMonth` (630)

Listing: `./examples/datutex/ex106.pp`

Program Example105;

{ This program demonstrates the DecodeDayOfWeekInMonth function }

Uses SysUtils, DateUtils;

Var

Y, M, NDoW, DoW : Word;

D : TDateTime;

Begin

`DecodeDayOfWeekInMonth(Date, Y, M, NDoW, DoW);`

If `TryEncodeDayOfWeekInMonth(Y, M, NDoW, DoW, D)` **then**

begin

`Write(DateToStr(D), ' is the ', NDoW, '-th ');`

`WriteLn(formatDateTime('dddd', D), ' of the month.');`

end

else

`WriteLn('Invalid year/month/NthDayOfWeek combination');`

End.

44.4.146 TryEncodeTimeInterval

Synopsis: Try to encode an interval as a `TDateTime` value.

Declaration: `function TryEncodeTimeInterval(Hour: Word; Min: Word; Sec: Word; MSec: Word; out Time: TDateTime) : Boolean`

Visibility: default

Description: `TryEncodeTimeInterval` encodes a time interval expressed in `Hour`, `Min`, `Sec`, `MSec` as a `TDateTime` value and returns the value in `Time`. It returns `True` if `Min`, `Sec`, `MSec` contain valid time values (i.e. less than 60, 60 resp. `MSec`). The number of hours may be larger than 24.

See also: `EncodeTimeInterval` (630)

44.4.147 TryISO8601ToDate

Synopsis: Attempts to convert an ISO 8601-formatted date/time value to a `TDateTime` type.

Declaration: `function TryISO8601ToDate(const DateString: string; out ADateTime: TDateTime; ReturnUTC: Boolean) : Boolean`

Visibility: default

Description: `TryISO8601ToDate` is a Boolean function which attempts to convert an ISO 8601-formatted date/time value to a `TDateTime` type.

`DateString` contains the ISO 8601 date/time value converted in the function. `DateString` must contain one of the supported ISO 8601 date/time notations supported in the routine. It is separated into date, time, and time zone values, and is converted by calling both the `TryISOStrToDateTime` and `TryISO TZStrToTZOffset` functions. See `TryISOStrToDateTime` (690) for more information about supported ISO 8601 notations.

`ReturnUTC` indicates if the native date/time value needs to be adjusted to the UTC (Coordinated Universal Time) time zone. The default value for the argument is `True`, and causes `GetLocalTimeOffset` to be called to get and apply the time zone offset for the local computer.

`ADateTime` contains the native `TDateTime` value for the converted timestamp adjusted to the UTC time zone when `ReturnUTC` contains `True`.

Use `DateToISO8601` to convert a native date/time value back to its representation in ISO 8601 notation.

See also: `DateToISO8601` (619)

44.4.148 TryISOStrToDate

Synopsis: Attempts to convert an ISO 8601-formatted date value to a `TDateTime` type.

Declaration: `function TryISOStrToDate(const aString: string; out outDate: TDateTime) : Boolean`

Visibility: default

Description: `TryISOStrToDate` is a Boolean function which attempts to convert an ISO 8601-formatted date value in `aString` to a `TDateTime` type. The return value is `True` if the string is successfully converted into a native date value.

`aString` contains the date value converted in the function, and can use one of the following ISO 8601 notations :

- `YYYY` (assumes month is 1 and day of month is 1)
- `YYYYMM` (assumes day of month is 1)
- `YYYY-MM` (assumes day of month is 1)
- `YYYYMMDD`
- `YYYY-MM-DD`

`outDate` is an output parameter where the converted `TDateTime` value is stored in the function. The return value is `False` (and `outDate` is set to an empty date value) if `aString` cannot be converted in the routine. The time portion of the `TDateTime` value is not used or updated in the function.

Use `TryISOStrToTime` to convert a time value to a native `TDateTime` type.

Use `TryISOStrToDateTime` to convert a string which contains both date and time values to a `TDateTime` type.

See also: `TryISOStrToTime` (691), `TryISOStrToDateTime` (690)

44.4.149 TryISOStrToDateTime

Synopsis: Attempts to convert an ISO 8601-formatted date/time value to a `TDateTime` type.

Declaration: `function TryISOStrToDateTime(const aString: string;
out outDateTime: TDateTime) : Boolean`

Visibility: default

Description: `TryISOStrToDateTime` is a `Boolean` function which attempts to convert an ISO 8601-formatted date/time value to a `TDateTime` type. `aString` contains the date/time value examined in the routine, and can use one of the supported ISO 8601 notations. Internally, `TryISOStrToDateTime` separates the value in `aString` into date and time parts and calls both `TryISOStrToDate` and `TryISOStrToTime`.

`outDateTime` is a `TDateTime` output parameter where the date/time value is stored in the function.

The date/time value examined in the routine can use one of the supported ISO 8601 notations. For example:

`[DATE] [SEPARATOR] [TIME] [TIMEZONE]`

where **[DATE]** can be one of the following formats:

- `YYYY` (assumes month is 1 and day of month is 1)
- `YYYYMM` (assumes day of month is 1)
- `YYYYMMDD` (eg 20210827)
- `YYYY-MM` (assumes day of month is 1)
- `YYYY-MM-DD` (eg 2021-08-27)

[SEPARATOR] can contain the following:

- **T** (literal character that marks the start of the time element). The **Space** character (**#32**) is also allowed as the separator at the start of the time value.

[TIME] can use one of the following formats:

- `hhmm`
- `hh:mm`
- `hhmmss`
- `hh:mm:ss`
- `hhmmss.zzz` (arbitrary number of digits for fractions of a second)
- `hh:mm:ss.zzz` (arbitrary number of digits for fractions of a second)

A time value with a fractional number of seconds can use either the **.'** (**Period**) or **','** (**Comma**) character for the decimal point.

[TIMEZONE] is an optional time zone designation, and uses one of the following notations:

- **Z** (Time zone Zulu or UTC)
- **±hh** (Positive or negative offset hours)
- **±hhmm** (Positive or negative offset hours and minutes)
- **±hh:mm** (Positive or negative offset hours and minutes)

In the above, the following formats are used:

YYYY Four digits for year

MM Two digits for month (01=January, etc.)

DD Two digits for day of month (01 through 31)

T Literal character used as a marker for the time value

hh Two digits for hour (00 through 23) (am/pm NOT allowed)

mm Two digits for minute (00 through 59)

ss Two digits for seconds (00 through 59)

z One or more digits representing the fractional seconds value (no maximum precision limit)

The return value is `True` if `aString` is successfully parsed and converted to a `TDateTime` type. The return value is `False` if `aString` contains a value that cannot be parsed in the function. When the return value is `False`, `outDateTime` is set to `0` (representing an empty date/time value).

Use `TryISOStrToDate` or `TryISOStrToTime` to convert a string using only a date or a time value (respectively).

Use `TryISO8601ToDate` to convert a string value which uses more intricate forms of the ISO 8601 time notation.

See also: `TryISOStrToDate` ([689](#)), `TryISOStrToTime` ([691](#)), `TryISO8601ToDate` ([688](#))

44.4.150 TryISOStrToTime

Synopsis: Converts an ISO 8601-formatted time value to a `TDateTime` type.

Declaration: `function TryISOStrToTime(const aString: string; out outTime: TDateTime)
: Boolean`

Visibility: default

Description: `TryISOStrToTime` is a `Boolean` function which attempts to convert the specified ISO 8601 time value to a `TDateTime` type. `aString` contains the ISO 8601 time value examined in the function, and can use one of the following notations:

- `hh` (hours)
- `hhnn` (hours and minutes)
- `hhnnss` (hours, minutes, seconds)
- `hhnnss.zzzz` (arbitrary number of decimal places for fractions of a second)
- `hhnnss.zzzzZ` (arbitrary precision and Zulu time zone designation)
- `hhnnss.zzzz±hh` (timezone offset in hours)
- `hhnnss.zzzz±hhnn` (timezone offset in hours and minutes)
- `hhnnss.zzzz±hh:nn` (timezone offset in hours and minutes)
- `hh:nn` (hours and minutes)
- `hh:nn:ss` (hours, minutes, seconds)
- `hh:nn:ss.zzz` (arbitrary number of decimal places for fractions of a second)
- `hh:nn:ss.zzzZ` (arbitrary precision and Zulu time zone designation)
- `hh:nn:ss.zzz±hh` (timezone offset in hours)
- `hh:nn:ss.zzz±hhnn` (timezone offset in hours and minutes)

- hh:nn:ss.zzz±hh:nn (timezone offset in hours and minutes)

aString may contain an optional time zone designation at the end of the string value, like '**Z**' for Zulu time zone or a time zone offset expressed using either positive or negative hours and minutes. For example:

- 120000Z
- 080000-04
- 080000-0400
- 080000-04:00
- 12:00:00Z
- 08:00:00-04
- 08:00:00-0400
- 08:00:00-04:00

The return value is `True` if the time value in aString is successfully converted in the function.

outTime is a `TDatetime` type used to store the time value converted in the function. When the return value is `False`, outTime contains `0` for an empty time value.

Use `TryISOStrToDate` to convert an ISO 8601 date value to a `TDatetime` type.

Use `TryISOStrToDateTime` to convert a string which contains both date and time values to a `TDatetime` type.

See also: `TryISOStrToDate` (689), `TryISOStrToDateTime` (690)

44.4.151 TryISOTZStrToTZOffset

Synopsis: Attempts to convert an ISO 8601 time zone designation to an offset in minutes.

Declaration: `function TryISOTZStrToTZOffset(const TZ: string; out TZOffset: Integer)
: Boolean`

Visibility: default

Description: `TryISOTZStrToTZOffset` is a `Boolean` function which attempts to convert the ISO 8601 time zone designation to a time zone offset expressed as a positive or negative number of minutes. `TZ` contains the time zone designation examined in the function. It should not contain any other portion of a date/time value using ISO 8601 notation - just the time zone designation.

`TryISOTZStrToTZOffset` recognizes the following values in the `TZ` argument:

Z Represents time zone Zulu (short for "Zulu time") as used by the military and in navigation. Refers to UTC (Coordinated Universal Time), formerly known as Greenwich Mean Time.

±HHNN or ±HH:NN time zone designation expressed as a positive or negative number of hours and minutes for a given time value.

`TZOffset` is an `Integer` updated to contains the number of minutes needed to adjust a time value for a given time zone back to UTC (Zulu time). The hour and minute component values in `TZ` are converted to minutes and complemented (multiplied by -1) to derive the offset value. The derived value in `TZOffset` is 0 when `TZ` contains Zulu time, an empty string (''), or cannot be converted successfully.

The return value is `True` if the time zone designation in `TZ` is successfully converted to a time zone offset.

Use `ISOTZStrToTZOffset` to convert the time zone designation and raise an exception for an invalid time zone designation.

See also: `ISOTZStrToTZOffset` (610)

44.4.152 TryJulianDateToDateTime

Synopsis: Convert a Julian date representation to a `TDateTime` value.

Declaration:

```
function TryJulianDateToDateTime(const AValue: Double;
                                out ADateTime: TDateTime) : Boolean
```

Visibility: default

Description: Try to convert a Julian date to a regular `TDateTime` date/time representation.

See also: [DateTimeToJulianDate \(618\)](#), [JulianDateToDateTime \(652\)](#), [DateTimeToModifiedJulianDate \(619\)](#), [TryModifiedJulianDateToDateTime \(693\)](#)

44.4.153 TryModifiedJulianDateToDateTime

Synopsis: Convert a modified Julian date representation to a `TDateTime` value.

Declaration:

```
function TryModifiedJulianDateToDateTime(const AValue: Double;
                                         out ADateTime: TDateTime)
                                         : Boolean
```

Visibility: default

Description: Convert a modified Julian date representation to a `TDateTime` value.

See also: [DateTimeToJulianDate \(618\)](#), [JulianDateToDateTime \(652\)](#), [TryJulianDateToDateTime \(693\)](#), [DateTimeToModifiedJulianDate \(619\)](#), [ModifiedJulianDateToDateTime \(661\)](#)

44.4.154 TryRecodeDateTime

Synopsis: Replace selected parts of a `TDateTime` value with other values.

Declaration:

```
function TryRecodeDateTime(const AValue: TDateTime; const AYear: Word;
                           const AMonth: Word; const ADay: Word;
                           const AHour: Word; const AMinute: Word;
                           const ASecond: Word;
                           const AMilliSecond: Word;
                           out AResult: TDateTime) : Boolean
```

Visibility: default

Description: `TryRecodeDateTime` replaces selected parts of the timestamp `AValue` with the date/time values specified in `AYear`, `AMonth`, `ADay`, `AHour`, `AMinute`, `ASecond` and `AMilliSecond`. If any of these values equals the predefined constant [RecodeLeaveFieldAsIs \(613\)](#), then the corresponding part of the date/time stamp is left untouched.

The resulting date/time is returned in `AValue`.

The function returns `True` if the encoding was successful. It returns `False` if one of the values `AYear`, `AMonth`, `ADay`, `AHour`, `AMinute`, `ASecond` or `AMilliSecond` is not within a valid range.

See also: [RecodeYear \(671\)](#), [RecodeMonth \(669\)](#), [RecodeDay \(666\)](#), [RecodeHour \(667\)](#), [RecodeMinute \(668\)](#), [RecodeSecond \(670\)](#), [RecodeMilliSecond \(668\)](#), [RecodeDate \(665\)](#), [RecodeTime \(670\)](#), [RecodeDateTime \(665\)](#)

Listing: `./examples/datutex/ex97.pp`

Program Example97;

{ This program demonstrates the TryRecodeDateTime function }

Uses SysUtils,DateUtils;

Const

Fmt = 'dddd dd mmmm yyyy hh:nn:ss';

Var

S : AnsiString;

D : TDateTime ;

Begin

If TryRecodeDateTime(**Now**,2000,2,RecodeLeaveFieldAsIs,0,0,0,D) **then**

begin

S:=**FormatDateTime**(Fmt,D);

WriteLn('This moment in februari 2000 : ',S);

end

else

WriteLn('This moment did/does not exist in februari 2000');

End.

44.4.155 UniversalTimeToLocal

Synopsis: Convert UTC time to local time.

Declaration: function UniversalTimeToLocal(UT: TDateTime) : TDateTime
function UniversalTimeToLocal(UT: TDateTime; TZOffset: Integer)
: TDateTime

Visibility: default

Description: UniversalTimeToLocal converts a universal time indication to a local time: it applies the TZOffset time zone offset to the UT Universal time (UTC). If no TZOffset is specified, the local time offset as returned by GetLocalTimeOffset (610) is used.

Note that for times in the past or in the future, or for time zones with DST, omitting the TZOffset may lead to wrong results depending on GetLocalTimeOffset being able to compute the correct offset for the LT on the target platform. Currently only Windows Vista and newer return correct offsets for a given date. Older Windows systems or Linux/Unix return always the offset for the current date.

See also: GetLocalTimeOffset (610), LocalTimeToUniversal (652)

44.4.156 UnixTimeStampToMac

Synopsis: Convert Unix Timestamp to a Mac Timestamp.

Declaration: function UnixTimeStampToMac(const AValue: Int64) : Int64

Visibility: default

Description: UnixTimeStampToMac converts the UNIX epoch time in AValue to a valid Mac timestamp indication and returns the result.

Errors: None.

See also: DateTimeToMac (618), MacToDateTime (653), MacTimeStampToUnix (652)

44.4.157 UnixToDateTime

Synopsis: Convert Unix epoch time to a `TDateTime` value.

Declaration: `function UnixToDateTime(const AValue: Int64; aReturnUTC: Boolean)
: TDateTime`

Visibility: default

Description: `UnixToDateTime` converts epoch time (seconds elapsed since 1/1/1970) to a `TDateTime` value.

See also: `DateTimeToUnix` (619)

44.4.158 WeekOf

Synopsis: Extract week (of the year) from a given date.

Declaration: `function WeekOf(const AValue: TDateTime) : Word`

Visibility: default

Description: `WeekOf` returns the week-of-the-year part of the `AValue` date/time indication. It is a number between 1 and 53.

For an example, see `YearOf` (707)

See also: `YearOf` (707), `DayOf` (620), `MonthOf` (661), `HourOf` (635), `MinuteOf` (657), `SecondOf` (675), `MilliSecondOf` (653)

44.4.159 WeekOfTheMonth

Synopsis: Extract the week of the month (and optionally month and year) from a `TDateTime` value.

Declaration: `function WeekOfTheMonth(const AValue: TDateTime) : Word; Overload
function WeekOfTheMonth(const AValue: TDateTime; out AYear: Word;
out AMonth: Word) : Word; Overload`

Visibility: default

Description: `WeekOfTheMonth` extracts the week of the month from `AValue` and returns it, and optionally returns the year and month as well (in `AYear`, `AMonth` respectively).

Remark Note that weeks are numbered from 1 using the ISO 8601 standard, and the day of the week as well. This means that the year and month may not be the same as the year part of the date, since the week may start in the previous year as the first week of the year is the week with at least 4 days in it.

See also: `WeekOfTheYear` (696), `DayOfTheMonth` (620), `HourOfTheMonth` (636), `MinuteOfTheMonth` (658), `SecondOfTheMonth` (676), `MilliSecondOfTheMonth` (654)

Listing: `./examples/datutex/ex41.pp`

Program Example41;

{ This program demonstrates the WeekOfTheMonth function }

Uses SysUtils, DateUtils;

Var

N : TDateTime;

```

Begin
  N:=Now;
  Writeln('Week of the Month : ',WeekOfTheMonth(N));
  Writeln('Day of the Month : ',DayOfTheMonth(N));
  Writeln('Hour of the Month : ',HourOfTheMonth(N));
  Writeln('Minute of the Month : ',MinuteOfTheMonth(N));
  Writeln('Second of the Month : ',SecondOfTheMonth(N));
  Writeln('MilliSecond of the Month : ',
    MilliSecondOfTheMonth(N));
End.

```

44.4.160 WeekOfTheYear

Synopsis: Extract the week of the year (and optionally year) of a TDateTime indication.

Declaration: `function WeekOfTheYear(const AValue: TDateTime) : Word; Overload`
`function WeekOfTheYear(const AValue: TDateTime; out AYear: Word) : Word`
`; Overload`

Visibility: default

Description: `WeekOfTheYear` extracts the week of the year from `AValue` and returns it, and optionally returns the year as well. It returns the same value as `WeekOf` (695).

Remark Note that weeks are numbered from 1 using the ISO 8601 standard, and the day of the week as well. This means that the year may not be the same as the year part of the date, since the week may start in the previous year as the first week of the year is the week with at least 4 days in it.

See also: `WeekOf` (695), `MonthOfTheYear` (661), `DayOfTheYear` (621), `HourOfTheYear` (637), `MinuteOfTheYear` (659), `SecondOfTheYear` (677), `MilliSecondOfTheYear` (655)

Listing: `./examples/datutex/ex40.pp`

Program Example40;

{ This program demonstrates the WeekOfTheYear function }

Uses SysUtils, DateUtils;

Var
 N : TDateTime;

```

Begin
  N:=Now;
  Writeln('Month of the year : ',MonthOfTheYear(N));
  Writeln('Week of the year : ',WeekOfTheYear(N));
  Writeln('Day of the year : ',DayOfTheYear(N));
  Writeln('Hour of the year : ',HourOfTheYear(N));
  Writeln('Minute of the year : ',MinuteOfTheYear(N));
  Writeln('Second of the year : ',SecondOfTheYear(N));
  Writeln('MilliSecond of the year : ',
    MilliSecondOfTheYear(N));
End.

```

44.4.161 WeeksBetween

Synopsis: Calculate the number of whole weeks between two TDateTime values.

Declaration: `function WeeksBetween(const ANow: TDateTime; const AThen: TDateTime)
: Integer`

Visibility: default

Description: `WeeksBetween` returns the number of whole weeks between `ANow` and `AThen`. This means the fractional part of a Week is dropped.

See also: `YearsBetween` (708), `MonthsBetween` (662), `DaysBetween` (621), `HoursBetween` (637), `MinutesBetween` (659), `SecondsBetween` (677), `MillisecondsBetween` (656)

Listing: `./examples/datutex/ex57.pp`

Program Example57;

{ This program demonstrates the WeeksBetween function }

Uses SysUtils, DateUtils;

Procedure Test(ANow, AThen : TDateTime);

begin

Write('Number of weeks between ');

Write(**DateToStr**(AThen), ' and ', **DateToStr**(ANow));

WriteLn(' : ', WeeksBetween(ANow, AThen));

end;

Var

 D1, D2 : TDateTime;

Begin

 D1 := Today;

 D2 := Today - 7;

 Test(D1, D2);

 D2 := Today - 8;

 Test(D1, D2);

 D2 := Today - 14;

 Test(D1, D2);

 D2 := Today - 35;

 Test(D1, D2);

 D2 := Today - 36;

 Test(D1, D2);

 D2 := Today - 17;

 Test(D1, D2);

End.

44.4.162 WeeksInAYear

Synopsis: Returns the number of weeks in a given year.

Declaration: `function WeeksInAYear(const AYear: Word) : Word`

Visibility: default

Description: `WeeksInYear` returns the number of weeks in the year `AYear`. The return value is either 52 or 53.

Remark The first week of the year is determined according to the ISO 8601 standard: It is the first week that has at least 4 days in it, i.e. it includes a Thursday.

See also: `WeeksInYear` (698), `DaysInYear` (624), `DaysInAYear` (623), `DaysInMonth` (623), `DaysInAMonth` (622)

Listing: `./examples/datutex/ex13.pp`

Program Example13;

{ This program demonstrates the WeeksInYear function }

Uses SysUtils, DateUtils;

Var

Y : Word;

Begin

For Y:=1992 to 2010 do

WriteLn(Y, ' has ', WeeksInYear(Y), ' weeks.');

End.

44.4.163 WeeksInYear

Synopsis: Returns the number of weeks in the year, given a date.

Declaration: `function WeeksInYear(const AValue: TDateTime) : Word`

Visibility: default

Description: `WeeksInYear` returns the number of weeks in the year part of `AValue`. The return value is either 52 or 53.

Remark The first week of the year is determined according to the ISO 8601 standard: It is the first week that has at least 4 days in it, i.e. it includes a Thursday.

See also: `WeeksInAYear` (697), `DaysInYear` (624), `DaysInAYear` (623), `DaysInMonth` (623), `DaysInAMonth` (622)

Listing: `./examples/datutex/ex12.pp`

Program Example12;

{ This program demonstrates the WeeksInYear function }

Uses SysUtils, DateUtils;

Var

Y : Word;

Begin

For Y:=1992 to 2010 do

WriteLn(Y, ' has ', WeeksInYear(EncodeDate(Y,2,1)), ' weeks.');

End.

44.4.164 WeekSpan

Synopsis: Calculate the approximate number of weeks between two TDateTime values.

Declaration: `function WeekSpan(const ANow: TDateTime; const AThen: TDateTime)
: Double`

Visibility: default

Description: `WeekSpan` returns the number of weeks between `ANow` and `AThen`, including any fractional parts of a week.

See also: `YearSpan` (709), `MonthSpan` (663), `DaySpan` (624), `HourSpan` (638), `MinuteSpan` (660), `SecondSpan` (678), `MilliSecondSpan` (656), `WeeksBetween` (697)

Listing: `./examples/datutex/ex65.pp`

Program Example57;

{ This program demonstrates the WeekSpan function }

Uses SysUtils, DateUtils;

Procedure Test(ANow, AThen : TDateTime);

begin

Write('Number of weeks between ');
 Write(**DateToStr**(AThen), ' and ', **DateToStr**(ANow));
 WriteLn(' : ', WeekSpan(ANow, AThen));
end;

Var

 D1, D2 : TDateTime;

Begin

 D1 := Today;
 D2 := Today - 7;
 Test(D1, D2);
 D2 := Today - 8;
 Test(D1, D2);
 D2 := Today - 14;
 Test(D1, D2);
 D2 := Today - 35;
 Test(D1, D2);
 D2 := Today - 36;
 Test(D1, D2);
 D2 := Today - 17;
 Test(D1, D2);

End.

44.4.165 WithinPastDays

Synopsis: Check whether two TDateTimes are only a number of days apart.

Declaration: `function WithinPastDays(const ANow: TDateTime; const AThen: TDateTime;
const ADays: Integer) : Boolean`

Visibility: default

Description: `WithinPastDays` compares the timestamps `ANow` and `AThen` and returns `True` if the difference between them is at most `ADays` days apart, or `False` if they are further apart.

Remark Since this function uses the `DaysBetween` (621) function to calculate the difference in days, this means that fractional days do not count, and the fractional part is simply dropped, so for two dates actually 2 and a half days apart, the result will also be `True`

See also: `WithinPastYears` (706), `WithinPastMonths` (703), `WithinPastWeeks` (705), `WithinPastHours` (700), `WithinPastMinutes` (702), `WithinPastSeconds` (704), `WithinPastMilliseconds` (701)

Listing: `./examples/datutex/ex50.pp`

Program Example50;

{ This program demonstrates the WithinPastDays function }

Uses SysUtils, DateUtils;

Procedure Test(`ANow, AThen` : `TDateTime`; `ADays` : `Integer`);

begin

`Write(DateTimeToStr(AThen), ' and ', DateTimeToStr(ANow));`

`Write(' are within ', ADays, ' days: ');`

`WriteLn(WithinPastDays(ANow, AThen, ADays));`

end;

Var

`D1, D2` : `TDateTime`;

Begin

`D1 := Now;`

`D2 := Today - 23/24;`

`Test(D1, D2, 1);`

`D2 := Today - 1;`

`Test(D1, D2, 1);`

`D2 := Today - 25/24;`

`Test(D1, D2, 1);`

`D2 := Today - 26/24;`

`Test(D1, D2, 5);`

`D2 := Today - 5.4;`

`Test(D1, D2, 5);`

`D2 := Today - 2.5;`

`Test(D1, D2, 1);`

`Test(D1, D2, 2);`

`Test(D1, D2, 3);`

End.

44.4.166 WithinPastHours

Synopsis: Check whether two `TDateTimes` are only a number of hours apart.

Declaration: `function WithinPastHours(const ANow: TDateTime; const AThen: TDateTime;
const AHours: Int64) : Boolean`

Visibility: default

Description: `WithinPastHours` compares the timestamps `ANow` and `AThen` and returns `True` if the difference between them is at most `AHours` hours apart, or `False` if they are further apart.

Remark Since this function uses the `HoursBetween` (637) function to calculate the difference in Hours, this means that fractional hours do not count, and the fractional part is simply dropped, so for two dates actually 2 and a half hours apart, the result will also be `True`

See also: `WithinPastYears` (706), `WithinPastMonths` (703), `WithinPastWeeks` (705), `WithinPastDays` (699), `WithinPastMinutes` (702), `WithinPastSeconds` (704), `WithinPastMilliseconds` (701)

Listing: `./examples/datutex/ex51.pp`

Program `Example51;`

{ This program demonstrates the WithinPastHours function }

Uses `SysUtils,DateUtils;`

Procedure `Test(ANow,AThen : TDateTime; AHours : Integer);`

begin
`Write(DateTimeToStr(AThen),' and ',DateTimeToStr(ANow));`
`Write(' are within ',AHours,' hours: ');`
`WriteLn(WithinPastHours(ANow,AThen,AHours));`
end;

Var

`D1,D2 : TDateTime;`

Begin

`D1:=Now;`
`D2:=D1-(59*OneMinute);`
`Test(D1,D2,1);`
`D2:=D1-(61*OneMinute);`
`Test(D1,D2,1);`
`D2:=D1-(122*OneMinute);`
`Test(D1,D2,1);`
`D2:=D1-(306*OneMinute);`
`Test(D1,D2,5);`
`D2:=D1-(5.4*OneHour);`
`Test(D1,D2,5);`
`D2:=D1-(2.5*OneHour);`
`Test(D1,D2,1);`
`Test(D1,D2,2);`
`Test(D1,D2,3);`

End.

44.4.167 WithinPastMilliseconds

Synopsis: Check whether two `TDateTimes` are only a number of milliseconds apart.

Declaration: `function WithinPastMilliseconds(const ANow: TDateTime;
const AThen: TDateTime;
const AMilliseconds: Int64) : Boolean`

Visibility: `default`

Description: `WithinPastMilliseconds` compares the timestamps `ANow` and `AThen` and returns `True` if the difference between them is at most `AMilliseconds` milliseconds apart, or `False` if they are further apart.

Remark Since this function uses the `MillisecondsBetween` (656) function to calculate the difference in milliseconds, this means that fractional milliseconds do not count, and the fractional part is simply dropped, so for two dates actually 2 and a half milliseconds apart, the result will also be `True`

See also: `WithinPastYears` (706), `WithinPastMonths` (703), `WithinPastWeeks` (705), `WithinPastDays` (699), `WithinPastHours` (700), `WithinPastMinutes` (702), `WithinPastSeconds` (704)

Listing: `./examples/datutex/ex54.pp`

Program `Example54;`

{ This program demonstrates the `WithinPastMilliseconds` function }

Uses `SysUtils,DateUtils;`

Procedure `Test(ANow,AThen : TDateTime; AMilliseconds : Integer);`

```
begin
  Write(TimeToStr(AThen),' and ',TimeToStr(ANow));
  Write(' are within ',AMilliseconds,' milliseconds: ');
  WriteLn(WithinPastMilliseconds(ANow,AThen,AMilliseconds));
end;
```

Var

`D1,D2 : TDateTime;`

Begin

```
D1:=Now;
D2:=D1-(0.9*OneMilliSecond);
Test(D1,D2,1);
D2:=D1-(1.0*OneMilliSecond);
Test(D1,D2,1);
D2:=D1-(1.1*OneMilliSecond);
Test(D1,D2,1);
D2:=D1-(2.5*OneMilliSecond);
Test(D1,D2,1);
Test(D1,D2,2);
Test(D1,D2,3);
```

End.

44.4.168 WithinPastMinutes

Synopsis: Check whether two `TDateTimes` are only a number of minutes apart.

Declaration: `function WithinPastMinutes(const ANow: TDateTime;`
`const AThen: TDateTime; const AMinutes: Int64)`
`: Boolean`

Visibility: `default`

Description: `WithinPastMinutes` compares the timestamps `ANow` and `AThen` and returns `True` if the difference between them is at most `AMinutes` minutes apart, or `False` if they are further apart.

Remark Since this function uses the `MinutesBetween` (659) function to calculate the difference in Minutes, this means that fractional minutes do not count, and the fractional part is simply dropped, so for two dates actually 2 and a half minutes apart, the result will also be `True`

See also: [WithinPastYears \(706\)](#), [WithinPastMonths \(703\)](#), [WithinPastWeeks \(705\)](#), [WithinPastDays \(699\)](#), [WithinPastHours \(700\)](#), [WithinPastSeconds \(704\)](#), [WithinPastMilliseconds \(701\)](#)

Listing: ./examples/datutex/ex52.pp

Program Example52;

{ This program demonstrates the WithinPastMinutes function }

Uses SysUtils, DateUtils;

Procedure Test(ANow, AThen : TDateTime; AMinutes : Integer);

begin

Write(DateTimeToStr(AThen), ' and ', DateTimeToStr(ANow));

Write(' are within ', AMinutes, ' Minutes: ');

WriteLn(WithinPastMinutes(ANow, AThen, AMinutes));

end;

Var

 D1, D2 : TDateTime;

Begin

 D1 := **Now**;

 D2 := D1 - (59 * OneSecond);

 Test(D1, D2, 1);

 D2 := D1 - (61 * OneSecond);

 Test(D1, D2, 1);

 D2 := D1 - (122 * OneSecond);

 Test(D1, D2, 1);

 D2 := D1 - (306 * OneSecond);

 Test(D1, D2, 5);

 D2 := D1 - (5.4 * OneMinute);

 Test(D1, D2, 5);

 D2 := D1 - (2.5 * OneMinute);

 Test(D1, D2, 1);

 Test(D1, D2, 2);

 Test(D1, D2, 3);

End.

44.4.169 WithinPastMonths

Synopsis: Check whether two TDateTimes are only a number of months apart.

Declaration: function WithinPastMonths(const ANow: TDateTime;
const AThen: TDateTime; const AMonths: Integer)
: Boolean

Visibility: default

Description: WithinPastMonths compares the timestamps ANow and AThen and returns True if the difference between them is at most AMonths months apart, or False if they are further apart.

Remark Since this function uses the MonthsBetween (662) function to calculate the difference in Months, this means that fractional months do not count, and the fractional part is simply dropped, so for two dates actually 2 and a half months apart, the result will also be True

See also: [WithinPastYears \(706\)](#), [WithinPastWeeks \(705\)](#), [WithinPastDays \(699\)](#), [WithinPastHours \(700\)](#), [WithinPastMinutes \(702\)](#), [WithinPastSeconds \(704\)](#), [WithinPastMilliseconds \(701\)](#)

Listing: ./examples/datutex/ex48.pp

Program Example48;

{ This program demonstrates the WithinPastMonths function }

Uses SysUtils,DateUtils;

Procedure Test(ANow,AThen : TDateTime; AMonths : Integer);

begin

Write(DateToStr(AThen),' and ',DateToStr(ANow));

Write(' are within ',AMonths,' months: ');

WriteLn(WithinPastMonths(ANow,AThen,AMonths));

end;

Var

 D1,D2 : TDateTime;

Begin

 D1:=Today;

 D2:=Today-364;

 Test(D1,D2,12);

 D2:=Today-365;

 Test(D1,D2,12);

 D2:=Today-366;

 Test(D1,D2,12);

 D2:=Today-390;

 Test(D1,D2,12);

 D2:=Today-368;

 Test(D1,D2,11);

 D2:=Today-1000;

 Test(D1,D2,31);

 Test(D1,D2,32);

 Test(D1,D2,33);

End.

44.4.170 WithinPastSeconds

Synopsis: Check whether two TDateTimes are only a number of seconds apart.

Declaration: function WithinPastSeconds(const ANow: TDateTime;
 const AThen: TDateTime; const ASeconds: Int64)
 : Boolean

Visibility: default

Description: WithinPastSeconds compares the timestamps ANow and AThen and returns True if the difference between them is at most ASeconds seconds apart, or False if they are further apart.

Remark Since this function uses the SecondsBetween (677) function to calculate the difference in seconds, this means that fractional seconds do not count, and the fractional part is simply dropped, so for two dates actually 2 and a half seconds apart, the result will also be True

See also: [WithinPastYears \(706\)](#), [WithinPastMonths \(703\)](#), [WithinPastWeeks \(705\)](#), [WithinPastDays \(699\)](#), [WithinPastHours \(700\)](#), [WithinPastMinutes \(702\)](#), [WithinPastMilliseconds \(701\)](#)

Listing: ./examples/datutex/ex53.pp

Program Example53;

{ This program demonstrates the WithinPastSeconds function }

Uses SysUtils, DateUtils;

Procedure Test(ATimeNow, ATimeThen : TDateTime; ASeconds : Integer);

begin

Write(DateTimeToStr(ATimeThen), ' and ', DateTimeToStr(ATimeNow));

Write(' are within ', ASeconds, ' seconds: ');

WriteLn(WithinPastSeconds(ATimeNow, ATimeThen, ASeconds));

end;

Var

 D1, D2 : TDateTime;

Begin

 D1 := **Now**;

 D2 := D1 - (999 * OneMilliSecond);

 Test(D1, D2, 1);

 D2 := D1 - (1001 * OneMilliSecond);

 Test(D1, D2, 1);

 D2 := D1 - (2001 * OneMilliSecond);

 Test(D1, D2, 1);

 D2 := D1 - (5001 * OneMilliSecond);

 Test(D1, D2, 5);

 D2 := D1 - (5.4 * OneSecond);

 Test(D1, D2, 5);

 D2 := D1 - (2.5 * OneSecond);

 Test(D1, D2, 1);

 Test(D1, D2, 2);

 Test(D1, D2, 3);

End.

44.4.171 WithinPastWeeks

Synopsis: Check whether two TDateTimes are only a number of weeks apart.

Declaration: `function WithinPastWeeks(const ANow: TDateTime; const AThen: TDateTime;
 const AWeeks: Integer) : Boolean`

Visibility: default

Description: `WithinPastWeeks` compares the timestamps `ANow` and `AThen` and returns `True` if the difference between them is at most `AWeeks` weeks apart, or `False` if they are further apart.

Remark Since this function uses the `WeeksBetween (697)` function to calculate the difference in Weeks, this means that fractional Weeks do not count, and the fractional part is simply dropped, so for two dates actually 2 and a half weeks apart, the result will also be `True`

See also: [WithinPastYears \(706\)](#), [WithinPastMonths \(703\)](#), [WithinPastDays \(699\)](#), [WithinPastHours \(700\)](#), [WithinPastMinutes \(702\)](#), [WithinPastSeconds \(704\)](#), [WithinPastMilliseconds \(701\)](#)

Listing: ./examples/datutex/ex49.pp

Program Example49;

{ This program demonstrates the WithinPastWeeks function }

Uses SysUtils, DateUtils;

Procedure Test(ANow, AThen : TDateTime; AWeeks : Integer);

begin

Write(DateToStr(AThen), ' and ', DateToStr(ANow));

Write(' are within ', AWeeks, ' weeks: ');

WriteLn(WithinPastWeeks(ANow, AThen, AWeeks));

end;

Var

 D1, D2 : TDateTime;

Begin

 D1 := Today;

 D2 := Today - 7;

 Test(D1, D2, 1);

 D2 := Today - 8;

 Test(D1, D2, 1);

 D2 := Today - 14;

 Test(D1, D2, 1);

 D2 := Today - 35;

 Test(D1, D2, 5);

 D2 := Today - 36;

 Test(D1, D2, 5);

 D2 := Today - 17;

 Test(D1, D2, 1);

 Test(D1, D2, 2);

 Test(D1, D2, 3);

End.

44.4.172 WithinPastYears

Synopsis: Check whether two TDateTimes are only a number of years apart.

Declaration: function WithinPastYears(const ANow: TDateTime; const AThen: TDateTime;
 const AYears: Integer) : Boolean

Visibility: default

Description: WithinPastYears compares the timestamps ANow and AThen and returns True if the difference between them is at most AYears years apart, or False if they are further apart.

Remark Since this function uses the YearsBetween (708) function to calculate the difference in years, this means that fractional years do not count, and the fractional part is simply dropped, so for two dates actually 2 and a half years apart, the result will also be True

See also: WithinPastMonths (703), WithinPastWeeks (705), WithinPastDays (699), WithinPastHours (700), WithinPastMinutes (702), WithinPastSeconds (704), WithinPastMilliseconds (701)

Listing: ./examples/datutex/ex47.pp

Program Example47;

{ This program demonstrates the WithinPastYears function }

Uses SysUtils,DateUtils;

Procedure Test(ANow,AThen : TDateTime; AYears : Integer);

begin

Write(DateToStr(AThen),' and ',DateToStr(ANow));

Write(' are within ',AYears,' years: ');

WriteLn(WithinPastYears(ANow,AThen,AYears));

end;

Var

 D1,D2 : TDateTime;

Begin

 D1:=Today;

 D2:=Today-364;

 Test(D1,D2,1);

 D2:=Today-365;

 Test(D1,D2,1);

 D2:=Today-366;

 Test(D1,D2,1);

 D2:=Today-390;

 Test(D1,D2,1);

 D2:=Today-368;

 Test(D1,D2,1);

 D2:=Today-1000;

 Test(D1,D2,1);

 Test(D1,D2,2);

 Test(D1,D2,3);

End.

44.4.173 YearOf

Synopsis: Extract the year from a given date.

Declaration: `function YearOf(const AValue: TDateTime) : Word`

Visibility: default

Description: YearOf returns the year part of the AValue date/time indication. It is a number between 1 and 9999.

See also: MonthOf ([661](#)), DayOf ([620](#)), WeekOf ([695](#)), HourOf ([635](#)), MinuteOf ([657](#)), SecondOf ([675](#)), MilliSecondOf ([653](#))

Listing: ./examples/datutex/ex23.pp

Program Example23;

{ This program demonstrates the YearOf function }

Uses SysUtils,DateUtils;

```

Var
  D : TDateTime;

Begin
  D:=Now;
  Writeln('Year : ',YearOf(D));
  Writeln('Month : ',MonthOf(D));
  Writeln('Day : ',DayOf(D));
  Writeln('Week : ',WeekOf(D));
  Writeln('Hour : ',HourOf(D));
  Writeln('Minute : ',MinuteOf(D));
  Writeln('Second : ',SecondOf(D));
  Writeln('MilliSecond : ',MilliSecondOf(D));
End.

```

44.4.174 YearsBetween

Synopsis: Calculate the number of whole years between two TDateTime values.

Declaration: `function YearsBetween(const ANow: TDateTime; const AThen: TDateTime; AExact: Boolean) : Integer`

Visibility: default

Description: `YearsBetween` returns the number of whole years between `ANow` and `AThen`. This number is an approximation, based on an average number of days of 365.25 per year (average over 4 years). This means the fractional part of a year is dropped.

See also: `MonthsBetween` ([662](#)), `WeeksBetween` ([697](#)), `DaysBetween` ([621](#)), `HoursBetween` ([637](#)), `MinutesBetween` ([659](#)), `SecondsBetween` ([677](#)), `MillisecondsBetween` ([656](#)), `YearSpan` ([709](#))

Listing: `./examples/datutex/ex55.pp`

Program Example55;

{ This program demonstrates the YearsBetween function }

Uses SysUtils,DateUtils;

Procedure Test(ANow,AThen : TDateTime);

```

begin
  Write('Number of years between ');
  Write(DateToStr(AThen),' and ',DateToStr(ANow));
  Writeln(' : ',YearsBetween(ANow,AThen));
end;

```

```

Var
  D1,D2 : TDateTime;

```

```

Begin
  D1:=Today;
  D2:=Today-364;
  Test(D1,D2);
  D2:=Today-365;
  Test(D1,D2);
  D2:=Today-366;

```

```

Test(D1,D2);
D2:=Today-390;
Test(D1,D2);
D2:=Today-368;
Test(D1,D2);
D2:=Today-1000;
Test(D1,D2);
End.

```

44.4.175 YearSpan

Synopsis: Calculate the approximate number of years between two TDateTime values.

Declaration: `function YearSpan(const ANow: TDateTime; const AThen: TDateTime)
: Double`

Visibility: default

Description: `YearSpan` returns the number of years between `ANow` and `AThen`, including any fractional parts of a year. This number is an approximation, based on an average number of days of 365.25 per year (average over 4 years).

See also: `MonthSpan` (663), `WeekSpan` (699), `DaySpan` (624), `HourSpan` (638), `MinuteSpan` (660), `SecondSpan` (678), `MilliSecondSpan` (656), `YearsBetween` (708)

Listing: `./examples/datutex/ex63.pp`

Program Example63;

{ This program demonstrates the YearSpan function }

Uses SysUtils, DateUtils;

Procedure Test(ANow, AThen : TDateTime);

begin

```

Write('Number of years between ');
Write(DateToStr(AThen), ' and ', DateToStr(ANow));
Writeln(' : ', YearSpan(ANow, AThen));
end;

```

Var

```

D1, D2 : TDateTime;

```

Begin

```

D1:=Today;
D2:=Today-364;
Test(D1,D2);
D2:=Today-365;
Test(D1,D2);
D2:=Today-366;
Test(D1,D2);
D2:=Today-390;
Test(D1,D2);
D2:=Today-368;
Test(D1,D2);
D2:=Today-1000;

```

```
Test(D1,D2);  
End.
```

44.4.176 Yesterday

Synopsis: Return the previous day.

Declaration: `function Yesterday : TDateTime`

Visibility: default

Description: `Yesterday` returns yesterday's date. `Yesterday` is determined from the system clock, i.e. it is `Today` (684) -1.

See also: `Today` (684), `Tomorrow` (684)

Listing: `./examples/datutex/ex18.pp`

Program Example18;

{ This program demonstrates the Yesterday function }

Uses SysUtils,DateUtils;

Begin

WriteLn(**FormatDateTime**("Today is " dd mmmm yyyy',Today));

WriteLn(**FormatDateTime**("Yesterday was " dd mmmm yyyy',Yesterday));

End.

44.5 EDateTimeException

44.6 ELocalTimeInvalid

44.7 TDateTimeHelper

44.7.1 Method overview

Page	Method	Description
715	AddDays	
715	AddHours	
716	AddMilliseconds	
715	AddMinutes	
715	AddMonths	
716	AddSeconds	
715	AddYears	
716	CompareTo	
713	Create	
713	CreateLocal	
713	CreateTotalSeconds	
713	CreateUnixTime	
717	DaysBetween	
715	EndOfDay	
714	EndOfMonth	
715	EndOfWeek	
714	EndOfYear	
716	Equals	
714	Format	
714	GetDate	
714	GetTime	
717	HoursBetween	
716	InRange	
717	IsAM	
716	IsInLeapYear	
717	IsPM	
716	IsSameDay	
716	IsToday	
718	MillisecondsBetween	
717	MinutesBetween	
717	MonthsBetween	
717	SecondsBetween	
715	StartOfDay	
714	StartOfMonth	
715	StartOfWeek	
714	StartOfYear	
714	ToString	
717	WeeksBetween	
719	WithinDays	
719	WithinHours	
720	WithinMilliseconds	
719	WithinMinutes	
719	WithinMonths	
718	WithinPastDays	
718	WithinPastHours	
719	WithinPastMilliseconds	
718	WithinPastMinutes	
718	WithinPastMonths	
718	WithinPastSeconds	
718	WithinPastWeeks	
718	WithinPastYears	

44.7.2 Property overview

Page	Properties	Access	Description
720	Date	r	
721	Day	r	
721	DayOfWeek	r	
721	DayOfYear	r	
721	Hour	r	
722	Millisecond	r	
721	Minute	r	
721	Month	r	
720	Now	r	
722	Second	r	
720	Time	r	
720	Today	r	
720	Tomorrow	r	
722	TotalSeconds	r	
722	UnixTime	r	
721	Year	r	
720	Yesterday	r	

44.7.3 TDateTimeHelper.Create

Declaration: `class function Create(const aYear: Word; const aMonth: Word;
const aDay: Word) : TDateTime; Overload; Static`
`class function Create(const aYear: Word; const aMonth: Word;
const aDay: Word; const aHour: Word;
const aMinute: Word; const aSecond: Word;
const aMillisecond: Word) : TDateTime; Overload
; Static`
`class function Create(Date: string; aFormat: string;
aDateSeparator: Char; aTimeSeparator: Char)
: TDateTime; Overload; Static`

Visibility: public

44.7.4 TDateTimeHelper.CreateLocal

Declaration: `class function CreateLocal(Date: string; local: string) : TDateTime
; Static`

Visibility: public

44.7.5 TDateTimeHelper.CreateUnixTime

Declaration: `class function CreateUnixTime(const Value: Int64) : TDateTime; Static`

Visibility: public

44.7.6 TDateTimeHelper.CreateTotalSeconds

Declaration: `class function CreateTotalSeconds(const Value: Int64) : TDateTime
; Static`

Visibility: public

44.7.7 TDateTimeHelper.GetDate

Declaration: `function GetDate : TDateTime`

Visibility: public

44.7.8 TDateTimeHelper.GetTime

Declaration: `function GetTime : TDateTime`

Visibility: public

44.7.9 TDateTimeHelper.ToString

Declaration: `function ToString(const aFormatStr: string) : string`

Visibility: public

44.7.10 TDateTimeHelper.Format

Declaration: `function Format(const aFormatStr: string) : string`

Visibility: public

44.7.11 TDateTimeHelper.StartOfYear

Declaration: `function StartOfYear : TDateTime`

Visibility: public

44.7.12 TDateTimeHelper.EndOfYear

Declaration: `function EndOfYear : TDateTime`

Visibility: public

44.7.13 TDateTimeHelper.StartOfMonth

Declaration: `function StartOfMonth : TDateTime`

Visibility: public

44.7.14 TDateTimeHelper.EndOfMonth

Declaration: `function EndOfMonth : TDateTime`

Visibility: public

44.7.15 TDateTimeHelper.StartOfWeek

Declaration: `function StartOfWeek : TDateTime`

Visibility: public

44.7.16 TDateTimeHelper.EndOfWeek

Declaration: `function EndOfWeek : TDateTime`

Visibility: public

44.7.17 TDateTimeHelper.StartOfDay

Declaration: `function StartOfDay : TDateTime`

Visibility: public

44.7.18 TDateTimeHelper.EndOfDay

Declaration: `function EndOfDay : TDateTime`

Visibility: public

44.7.19 TDateTimeHelper.AddYears

Declaration: `function AddYears(const aNumberOfYears: Integer) : TDateTime`

Visibility: public

44.7.20 TDateTimeHelper.AddMonths

Declaration: `function AddMonths(const aNumberOfMonths: Integer) : TDateTime`

Visibility: public

44.7.21 TDateTimeHelper.AddDays

Declaration: `function AddDays(const aNumberOfDays: Integer) : TDateTime`

Visibility: public

44.7.22 TDateTimeHelper.AddHours

Declaration: `function AddHours(const aNumberOfHours: Int64) : TDateTime`

Visibility: public

44.7.23 TDateTimeHelper.AddMinutes

Declaration: `function AddMinutes(const aNumberOfMinutes: Int64) : TDateTime`

Visibility: public

44.7.24 TDateTimeHelper.AddSeconds

Declaration: `function AddSeconds(const aNumberOfSeconds: Int64) : TDateTime`

Visibility: public

44.7.25 TDateTimeHelper.AddMilliseconds

Declaration: `function AddMilliseconds(const aNumberOfMilliseconds: Int64) : TDateTime`

Visibility: public

44.7.26 TDateTimeHelper.CompareTo

Declaration: `function CompareTo(const aDateTime: TDateTime) : TValueRelationship`

Visibility: public

44.7.27 TDateTimeHelper.Equals

Declaration: `function Equals(const aDateTime: TDateTime) : Boolean`

Visibility: public

44.7.28 TDateTimeHelper.IsSameDay

Declaration: `function IsSameDay(const aDateTime: TDateTime) : Boolean`

Visibility: public

44.7.29 TDateTimeHelper.InRange

Declaration: `function InRange(const aStartDateTime: TDateTime;
 const aEndDateTime: TDateTime;
 const aInclusive: Boolean) : Boolean`

Visibility: public

44.7.30 TDateTimeHelper.IsInLeapYear

Declaration: `function IsInLeapYear : Boolean`

Visibility: public

44.7.31 TDateTimeHelper.IsToday

Declaration: `function IsToday : Boolean`

Visibility: public

44.7.32 TDateTimeHelper.IsAM

Declaration: `function IsAM : Boolean`

Visibility: `public`

44.7.33 TDateTimeHelper.IsPM

Declaration: `function IsPM : Boolean`

Visibility: `public`

44.7.34 TDateTimeHelper.YearsBetween

Declaration: `function YearsBetween(const aDateTime: TDateTime) : Integer`

Visibility: `public`

44.7.35 TDateTimeHelper.MonthsBetween

Declaration: `function MonthsBetween(const aDateTime: TDateTime) : Integer`

Visibility: `public`

44.7.36 TDateTimeHelper.WeeksBetween

Declaration: `function WeeksBetween(const aDateTime: TDateTime) : Integer`

Visibility: `public`

44.7.37 TDateTimeHelper.DaysBetween

Declaration: `function DaysBetween(const aDateTime: TDateTime) : Integer`

Visibility: `public`

44.7.38 TDateTimeHelper.HoursBetween

Declaration: `function HoursBetween(const aDateTime: TDateTime) : Int64`

Visibility: `public`

44.7.39 TDateTimeHelper.MinutesBetween

Declaration: `function MinutesBetween(const aDateTime: TDateTime) : Int64`

Visibility: `public`

44.7.40 TDateTimeHelper.SecondsBetween

Declaration: `function SecondsBetween(const aDateTime: TDateTime) : Int64`

Visibility: `public`

44.7.41 TDateTimeHelper.MilliSecondsBetween

Declaration: `function MilliSecondsBetween(const aDateTime: TDateTime) : Int64`

Visibility: public

44.7.42 TDateTimeHelper.WithinPastYears

Declaration: `function WithinPastYears(const AThen: TDateTime; const AYears: Integer)
: Boolean`

Visibility: public

44.7.43 TDateTimeHelper.WithinPastMonths

Declaration: `function WithinPastMonths(const AThen: TDateTime;
const AMonths: Integer) : Boolean`

Visibility: public

44.7.44 TDateTimeHelper.WithinPastWeeks

Declaration: `function WithinPastWeeks(const AThen: TDateTime; const AWeeks: Integer)
: Boolean`

Visibility: public

44.7.45 TDateTimeHelper.WithinPastDays

Declaration: `function WithinPastDays(const AThen: TDateTime; const ADays: Integer)
: Boolean`

Visibility: public

44.7.46 TDateTimeHelper.WithinPastHours

Declaration: `function WithinPastHours(const AThen: TDateTime; const AHours: Int64)
: Boolean`

Visibility: public

44.7.47 TDateTimeHelper.WithinPastMinutes

Declaration: `function WithinPastMinutes(const AThen: TDateTime;
const AMinutes: Int64) : Boolean`

Visibility: public

44.7.48 TDateTimeHelper.WithinPastSeconds

Declaration: `function WithinPastSeconds(const AThen: TDateTime;
const ASeconds: Int64) : Boolean`

Visibility: public

44.7.49 TDateTimeHelper.WithinPastMilliseconds

Declaration: `function WithinPastMilliseconds(const AThen: TDateTime;
const AMilliSeconds: Int64) : Boolean`

Visibility: public

44.7.50 TDateTimeHelper.WithinYears

Declaration: `function WithinYears(const aDateTime: TDateTime; const aYears: Integer)
: Boolean`

Visibility: public

44.7.51 TDateTimeHelper.WithinMonths

Declaration: `function WithinMonths(const aDateTime: TDateTime;
const aMonths: Integer) : Boolean`

Visibility: public

44.7.52 TDateTimeHelper.WithinWeeks

Declaration: `function WithinWeeks(const aDateTime: TDateTime; const aWeeks: Integer)
: Boolean`

Visibility: public

44.7.53 TDateTimeHelper.WithinDays

Declaration: `function WithinDays(const aDateTime: TDateTime; const aDays: Integer)
: Boolean`

Visibility: public

44.7.54 TDateTimeHelper.WithinHours

Declaration: `function WithinHours(const aDateTime: TDateTime; const aHours: Int64)
: Boolean`

Visibility: public

44.7.55 TDateTimeHelper.WithinMinutes

Declaration: `function WithinMinutes(const aDateTime: TDateTime;
const aMinutes: Int64) : Boolean`

Visibility: public

44.7.56 TDateTimeHelper.WithinSeconds

Declaration: `function WithinSeconds(const aDateTime: TDateTime;
const aSeconds: Int64) : Boolean`

Visibility: public

44.7.57 TDateTimeHelper.WithinMilliseconds

Declaration: `function WithinMilliseconds(const aDateTime: TDateTime;
const AMilliseconds: Int64) : Boolean`

Visibility: public

44.7.58 TDateTimeHelper.Now

Declaration: `Property Now : TDateTime`

Visibility: public

Access: Read

44.7.59 TDateTimeHelper.Today

Declaration: `Property Today : TDateTime`

Visibility: public

Access: Read

44.7.60 TDateTimeHelper.Yesterday

Declaration: `Property Yesterday : TDateTime`

Visibility: public

Access: Read

44.7.61 TDateTimeHelper.Tomorrow

Declaration: `Property Tomorrow : TDateTime`

Visibility: public

Access: Read

44.7.62 TDateTimeHelper.Date

Declaration: `Property Date : TDateTime`

Visibility: public

Access: Read

44.7.63 TDateTimeHelper.Time

Declaration: `Property Time : TDateTime`

Visibility: public

Access: Read

44.7.64 TDateTimeHelper.DayOfWeek

Declaration: `Property DayOfWeek : Word`

Visibility: public

Access: Read

44.7.65 TDateTimeHelper.DayOfYear

Declaration: `Property DayOfYear : Word`

Visibility: public

Access: Read

44.7.66 TDateTimeHelper.Year

Declaration: `Property Year : Word`

Visibility: public

Access: Read

44.7.67 TDateTimeHelper.Month

Declaration: `Property Month : Word`

Visibility: public

Access: Read

44.7.68 TDateTimeHelper.Day

Declaration: `Property Day : Word`

Visibility: public

Access: Read

44.7.69 TDateTimeHelper.Hour

Declaration: `Property Hour : Word`

Visibility: public

Access: Read

44.7.70 TDateTimeHelper.Minute

Declaration: `Property Minute : Word`

Visibility: public

Access: Read

44.7.71 TDateTimeHelper.SecondDeclaration: `Property Second : Word`Visibility: `public`Access: `Read`**44.7.72 TDateTimeHelper.Millisecond**Declaration: `Property Millisecond : Word`Visibility: `public`Access: `Read`**44.7.73 TDateTimeHelper.UnixTime**Declaration: `Property UnixTime : Int64`Visibility: `public`Access: `Read`**44.7.74 TDateTimeHelper.TotalSeconds**Declaration: `Property TotalSeconds : Int64`Visibility: `public`Access: `Read`**44.8 TTimeZone****44.8.1 Method overview**

Page	Method	Description
723	<code>Done</code>	
723	<code>GetAbbreviation</code>	
723	<code>GetDisplayName</code>	
724	<code>GetLocalTimeType</code>	
723	<code>GetUtcOffset</code>	
724	<code>HasDST</code>	
723	<code>Init</code>	
724	<code>IsAmbiguousTime</code>	
724	<code>IsDaylightTime</code>	
724	<code>IsInvalidTime</code>	
724	<code>IsStandardTime</code>	
723	<code>ToLocalTime</code>	
723	<code>ToUniversalTime</code>	

44.8.2 Property overview

Page	Properties	Access	Description
725	Abbreviation	r	
724	DisplayName	r	
724	ID	r	
725	Local	r	
725	UtcOffset	r	

44.8.3 TTimeZone.Init

Declaration: `class constructor Init`

Visibility: `public`

44.8.4 TTimeZone.Done

Declaration: `destructor Done`

Visibility: `public`

44.8.5 TTimeZone.GetUtcOffset

Declaration: `function GetUtcOffset(const aDateTime: TDateTime;
const aForceDaylight: Boolean) : TTimeSpan`

Visibility: `public`

44.8.6 TTimeZone.ToUniversalTime

Declaration: `function ToUniversalTime(const aDateTime: TDateTime;
const aForceDaylight: Boolean) : TDateTime`

Visibility: `public`

44.8.7 TTimeZone.ToLocalTime

Declaration: `function ToLocalTime(const aDateTime: TDateTime) : TDateTime`

Visibility: `public`

44.8.8 TTimeZone.GetDisplayName

Declaration: `function GetDisplayName(const aDateTime: TDateTime;
const aForceDaylight: Boolean) : string`

Visibility: `public`

44.8.9 TTimeZone.GetAbbreviation

Declaration: `function GetAbbreviation(const aDateTime: TDateTime;
const aForceDaylight: Boolean) : string`

Visibility: `public`

44.8.10 **TTimeZone.GetLocalTimeType**

Declaration: `function GetLocalTimeType(const aDateTime: TDateTime) : TLocalTimeType`

Visibility: public

44.8.11 **TTimeZone.HasDST**

Declaration: `function HasDST(const AYear: TDateTime) : Boolean; Overload`
`function HasDST : Boolean; Overload`

Visibility: public

44.8.12 **TTimeZone.IsStandardTime**

Declaration: `function IsStandardTime(const aDateTime: TDateTime;`
`const aForceDaylight: Boolean) : Boolean`

Visibility: public

44.8.13 **TTimeZone.IsInvalidTime**

Declaration: `function IsInvalidTime(const aDateTime: TDateTime) : Boolean`

Visibility: public

44.8.14 **TTimeZone.IsAmbiguousTime**

Declaration: `function IsAmbiguousTime(const aDateTime: TDateTime) : Boolean`

Visibility: public

44.8.15 **TTimeZone.IsDaylightTime**

Declaration: `function IsDaylightTime(const aDateTime: TDateTime;`
`const aForceDaylight: Boolean) : Boolean`

Visibility: public

44.8.16 **TTimeZone.ID**

Declaration: `Property ID : string`

Visibility: public

Access: Read

44.8.17 **TTimeZone.DisplayName**

Declaration: `Property DisplayName : string`

Visibility: public

Access: Read

44.8.18 TTimeZone.Abbreviation

Declaration: `Property Abbreviation : string`

Visibility: `public`

Access: `Read`

44.8.19 TTimeZone.UtcOffset

Declaration: `Property UtcOffset : TimeSpan`

Visibility: `public`

Access: `Read`

44.8.20 TTimeZone.Local

Declaration: `Property Local : TTimeZone`

Visibility: `public`

Access: `Read`

Chapter 45

Reference for unit 'Dos'

45.1 Used units

Table 45.1: Used units by unit 'Dos'

Name	Page
BaseUnix	155
System	1335

45.2 Overview

The DOS unit gives access to some operating system calls related to files, the file system, date and time. Except for the PalmOS target, this unit is available to all supported platforms.

The unit was first written for Dos by Florian Klaempfl. It was ported to Linux by Mark May and enhanced by Michael Van Canneyt. The Amiga version was ported by Nils Sjöholm.

Under non-DOS systems, some of the functionality is lost, as it is either impossible or meaningless to implement it. Other than that, the functionality is the same for all operating systems.

Because the DOS unit is a Turbo Pascal compatibility unit, it is no longer actively developed: the interface is frozen and it is maintained only for the purpose of porting Turbo Pascal programs. For new development, it is recommended to use the sysutils ([1609](#)) unit instead.

45.3 System information.

Functions for retrieving and setting general system information such as date and time.

Table 45.2:

Name	Description
DosVersion (735)	Get OS version
GetCBreak (740)	Get setting of control-break handling flag
GetDate (741)	Get system date
GetIntVec (743)	Get interrupt vector status
GetTime (745)	Get system time
GetVerify (745)	Get verify flag
Intr (746)	Execute an interrupt
Keep (746)	Keep process in memory and exit
MSDos (746)	Execute MS-Dos function call
PackTime (747)	Pack time for file time
SetCBreak (747)	Set control-break handling flag
SetDate (748)	Set system date
SetIntVec (749)	Set interrupt vectors
SetTime (749)	Set system time
SetVerify (749)	Set verify flag
SwapVectors (750)	Swap interrupt vectors
UnPackTime (750)	Unpack file time

45.4 Process handling.

Functions to handle process information and starting new processes.

Table 45.3:

Name	Description
DosExitCode (734)	Exit code of last executed program
EnvCount (736)	Return number of environment variables
EnvStr (736)	Return environment string pair
Exec (737)	Execute program
GetEnv (741)	Return specified environment string

45.5 Directory and disk handling.

Routines to handle disk information.

Table 45.4:

Name	Description
AddDisk (732)	Add disk to list of disks (UNIX only)
DiskFree (733)	Return size of free disk space
DiskSize (734)	Return total disk size

45.6 File handling.

Routines to handle files on disk.

Table 45.5:

Name	Description
FExpand (737)	Expand filename to full path
FindClose (737)	Close finfirst/findnext session
FindFirst (738)	Start find of file
FindNext (739)	Find next file
FSearch (739)	Search for file in a path
FSplit (740)	Split filename in parts
GetFAttr (742)	Return file attributes
GetFTime (743)	Return file time
GetLongName (744)	Convert short filename to long filename (DOS only)
GetShortName (744)	Convert long filename to short filename (DOS only)
SetFAttr (748)	Set file attributes
SetFTime (749)	Set file time

45.7 File open mode constants.

These constants are used in the Mode field of the TextRec record. Gives information on the file-mode of the text I/O. For their definitions consult the following table:

Table 45.6: Possible mode constants

Constant	Description	Value
fmclosed	File is closed.	\$D7B0
fminput	File is read only.	\$D7B1
fmoutput	File is write only.	\$D7B2
fminout	File is read and write.	\$D7B3

45.8 File attributes.

The File Attribute constants are used in FindFirst (738), FindNext (739) to determine what type of special file to search for in addition to normal files. These flags are also used in the SetFAttr (748) and GetFAttr (742) routines to set and retrieve attributes of files. For their definitions consult fileattributes (728).

Table 45.7: Possible file attributes

Constant	Description	Value
readonly	Read-Only file attribute.	\$01
hidden	Hidden file attribute.	\$02
sysfile	System file attribute.	\$04
volumeid	Volume ID file attribute.	\$08
directory	Directory file attribute.	\$10
archive	Archive file attribute.	\$20
anyfile	Match any file attribute.	\$3F

45.9 Constants, types and variables

45.9.1 Constants

`anyfile` = \$3F

Match any file attribute.

`archive` = \$20

Archive file attribute.

`directory` = \$10

Directory file attribute.

`fauxiliary` = \$0010

CPU auxiliary flag. Not used.

`fcarry` = \$0001

CPU carry flag. Not used.

`FileNameLen` = 255

Maximum length of a filename.

`fmclosed` = \$D7B0

File is closed.

`fminout` = \$D7B3

File is read and write.

`fminput` = \$D7B1

File is read only.

`fmoutput = $D7B2`

File is write only.

`foverflow = $0800`

CPU overflow flag. Not used.

`fparity = $0004`

CPU parity flag. Not used.

`fsign = $0080`

CPU sign flag. Not used.

`fzero = $0040`

CPU zero flag. Not used.

`hidden = $02`

Hidden file attribute.

`readonly = $01`

Read-Only file attribute.

`sysfile = $04`

System file attribute.

`volumeid = $08`

Volume ID file attribute.

45.9.2 Types

`ComStr = string`

Command-line string type.

`DirStr = string`

Full directory string type.

`ExtStr = string`

Filename extension string type.

```
NameStr = string
```

Fill filename string type.

```
PathStr = string
```

Full File path string type.

```
Registers = packed record
case i : Integer of
0: (
  ax : Word;
  f1 : Word;
  bx : Word;
  f2 : Word;
  cx : Word;
  f3 : Word;
  dx : Word;
  f4 : Word;
  bp : Word;
  f5 : Word;
  si : Word;
  f51 : Word;
  di : Word;
  f6 : Word;
  ds : Word;
  f7 : Word;
  es : Word;
  f8 : Word;
  flags : Word;
  fs : Word;
  gs : Word
  ;
);
1: (
  al : Byte;
  ah : Byte;
  f9 : Byte;
  f10 : Byte;
  bl
  : Byte;
  bh : Byte;
  f11 : Byte;
  f12 : Byte;
  cl : Byte;
  ch
  : Byte;
  f13 : Byte;
  f14 : Byte;
  dl : Byte;
  dh : Byte;
```

```
);
  2: (
    eax : LongInt;
    ebx : LongInt;
    ecx : LongInt;
    edx : LongInt
  ;
    ebp : LongInt;
    esi : LongInt;
    edi : LongInt;
  );
end
```

This structure is only defined on a i386 compatible 32-bit platform, and is not used anywhere: it is defined for Turbo Pascal backwards compatibility only.

45.9.3 Variables

```
DosError : Integer
```

The `DosError` variable is used by the procedures in the `Dos` unit to report errors. It can have the following values :

Table 45.8: Dos error codes

Value	Meaning
2	File not found.
3	Path not found.
5	Access denied.
6	Invalid handle.
8	Not enough memory.
10	Invalid environment.
11	Invalid format.
18	No more files.

Other values are possible, but are not documented.

45.10 Procedures and functions

45.10.1 AddDisk

Synopsis: Add disk definition to list if drives (Unix only).

Declaration: `function AddDisk(const path: string) : Byte`

Visibility: default

Description: `AddDisk` adds a filename `S` to the internal list of disks. It is implemented for systems which do not use DOS type drive letters. This list is used to determine which disks to use in the `DiskFree` (733) and `DiskSize` (734) calls. The `DiskFree` (733) and `DiskSize` (734) functions need a file on the specified drive, since this is required for the `statfs` system call. The names are added sequentially. The `Dos` initialization code presets the first three disks to:

- ' .' for the current drive,
- '/fd0/ .' for the first floppy-drive (Linux only).
- '/fd1/ .' for the second floppy-drive (Linux only).
- '/' for the first hard disk.

The first call to `AddDisk` will therefore add a name for the second harddisk, The second call for the third drive, and so on until 23 drives have been added (corresponding to drives 'D:' to 'Z:')

Errors: None

See also: [DiskFree \(733\)](#), [DiskSize \(734\)](#)

45.10.2 DiskFree

Synopsis: Get free size on Disk.

Declaration: `function DiskFree(drive: Byte) : Int64`

Visibility: default

Description: `DiskFree` returns the number of free bytes on a disk. The parameter `Drive` indicates which disk should be checked. This parameter is 1 for floppy a:, 2 for floppy b:, etc. A value of 0 returns the free space on the current drive.

Remark For Unices: The `diskfree` and `disksize` functions need a file on the specified drive, since this is required for the `statfs` system call. These filenames are set in the initialization of the `Dos` unit, and have been preset to :

- ' .' for the current drive,
- '/fd0/ .' for the first floppy-drive (Linux only).
- '/fd1/ .' for the second floppy-drive (Linux only).
- '/' for the first hard disk.

There is room for 1-26 drives. You can add a drive with the `AddDisk` ([732](#)) procedure. These settings can be coded in `dos.pp`, in the initialization part.

Errors: -1 when a failure occurs, or an invalid drive number is given.

See also: [DiskSize \(734\)](#), [AddDisk \(732\)](#)

Listing: `./examples/dosex/ex6.pp`

```

Program Example6;
uses Dos;

{ Program to demonstrate the DiskSize and DiskFree function. }

begin
  WriteLn('This partition size has ',DiskSize(0),' bytes');
  WriteLn('Currently ',DiskFree(0),' bytes are free');
end.
```

45.10.3 DiskSize

Synopsis: Get total size of disk.

Declaration: `function DiskSize(drive: Byte) : Int64`

Visibility: default

Description: `DiskSize` returns the total size (in bytes) of a disk. The parameter `Drive` indicates which disk should be checked. This parameter is 1 for floppy a:, 2 for floppy b:, etc. A value of 0 returns the size of the current drive.

Remark For Unix only: The `diskfree` and `disksize` functions need a file on the specified drive, since this is required for the `statfs` system call. These filenames are set in the initialization of the Dos unit, and have been preset to :

- ' .' for the current drive,
- '/fd0/ .' for the first floppy-drive (Linux only).
- '/fd1/ .' for the second floppy-drive (Linux only).
- '/' for the first hard disk.

There is room for 1-26 drives. You can add a drive with the `AddDisk` (732) procedure. These settings can be coded in `dos.pp`, in the initialization part.

For an example, see `DiskFree` (733).

Errors: -1 when a failure occurs, or an invalid drive number is given.

See also: `DiskFree` (733), `AddDisk` (732)

45.10.4 DosExitCode

Synopsis: Exit code of last executed program.

Declaration: `function DosExitCode : Word`

Visibility: default

Description: `DosExitCode` contains (in the low byte) the exit-code of a program executed with the `Exec` call.

Errors: None.

See also: `Exec` (737)

Listing: `./examples/dosex/ex5.pp`

```

Program Example5;
uses Dos;

{ Program to demonstrate the Exec and DosExitCode function. }

begin
  {$IFDEF Unix}
    WriteLn('Executing /bin/lis -la');
    Exec('/bin/lis', '-la');
  {$ELSE}
    WriteLn('Executing Dir');
    Exec(GetEnv('COMSPEC'), '/C dir');
  {$ENDIF}
  WriteLn('Program returned with ExitCode ', Lo(DosExitCode));
end.

```

45.10.5 DosVersion

Synopsis: Current OS version.

Declaration: `function DosVersion : Word`

Visibility: default

Description: `DosVersion` returns the operating system or kernel version. The low byte contains the major version number, while the high byte contains the minor version number.

Remark On systems where versions consists of more then two numbers, only the first two numbers will be returned. For example Linux version 2.1.76 will give you `DosVersion` 2.1. Some operating systems, such as FreeBSD, do not have system calls to return the kernel version, in that case a value of 0 will be returned.

Errors: None.

Listing: `./examples/dosex/ex1.pp`

```

Program Example1;
uses Dos;

{ Program to demonstrate the DosVersion function. }

var
  OS : string[32];
  Version : word;
begin
  {$IFDEF LINUX}
    OS:='Linux';
  {$ENDIF}
  {$ifdef FreeBSD}
    OS:='FreeBSD';
  {$endif}
  {$ifdef NetBSD}
    OS:='NetBSD';
  {$endif}
  {$ifdef Solaris}
    OS:='Solaris';
  {$endif}
  {$ifdef QNX}
    OS:='QNX';
  {$endif}

  {$IFDEF DOS}
    OS:='Dos';
  {$ENDIF}
  Version:=DosVersion;
  WriteLn('Current ',OS,' version is ',Lo(Version),', ',Hi(Version));
end.

```

45.10.6 DTToUnixDate

Synopsis: Convert a `DateTime` to Unix timestamp.

Declaration: `function DTToUnixDate(DT: DateTime) : LongInt`

Visibility: default

Description: `DTToUnixDate` converts the `DateTime` value in `DT` to a Unix timestamp. It is an internal function, implemented on Unix platforms, and should not be used.

Errors: None.

See also: `UnixDateToDT` (750), `PackTime` (747), `UnpackTime` (750), `GetTime` (745), `SetTime` (749)

45.10.7 EnvCount

Synopsis: Return the number of environment variables.

Declaration: `function EnvCount : LongInt`

Visibility: default

Description: `EnvCount` returns the number of environment variables.

Errors: None.

See also: `EnvStr` (736), `GetEnv` (741)

45.10.8 EnvStr

Synopsis: Return environment variable by index.

Declaration: `function EnvStr(Index: LongInt) : string`

Visibility: default

Description: `EnvStr` returns the `Index`-th `Name=Value` pair from the list of environment variables. The index of the first pair is zero.

Errors: The length is limited to 255 characters.

See also: `EnvCount` (736), `GetEnv` (741)

Listing: `./examples/dosex/ex13.pp`

```

Program Example13;
uses Dos;

{ Program to demonstrate the EnvCount and EnvStr function. }

var
  i : Longint;
begin
  WriteLn('Current Environment is:');
  for i:=1 to EnvCount do
    WriteLn(EnvStr(i));
end.

```

45.10.9 Exec

Synopsis: Execute another program, and wait for it to finish.

Declaration: `procedure Exec(const path: PathStr; const comline: ComStr)`

Visibility: default

Description: `Exec` executes the program in `Path`, with the options given by `ComLine`. The program name should *not* appear again in `ComLine`, it is specified in `Path`. `Comline` contains only the parameters that are passed to the program.

After the program has terminated, the procedure returns. The Exit value of the program can be consulted with the `DosExitCode` function.

For an example, see `DosExitCode` (734)

Errors: Errors are reported in `DosError`.

See also: `DosExitCode` (734)

45.10.10 FExpand

Synopsis: Expand a relative path to an absolute path.

Declaration: `function FExpand(const path: PathStr) : PathStr`

Visibility: default

Description: `FExpand` takes its argument and expands it to a complete filename, i.e. a filename starting from the root directory of the current drive, prepended with the drive-letter or volume name (when supported).

Remark On case sensitive file systems (such as Unix and Linux), the resulting name is left as it is, otherwise it is converted to uppercase.

Errors: `FSplit` (740)

Listing: `./examples/dosex/ex11.pp`

Program Example11;
uses Dos;

{ Program to demonstrate the FExpand function. }

begin

WriteLn('Expanded Name of this program is ',FExpand(**ParamStr**(0)));
end.

45.10.11 FindClose

Synopsis: Dispose resources allocated by a `FindFirst` (738)/`FindNext` (739) sequence.

Declaration: `procedure FindClose(var f: SearchRec)`

Visibility: default

Description: `FindClose` frees any resources associated with the search record `F`.

This call is needed to free any internal resources allocated by the `FindFirst` (738) or `FindNext` (739) calls.

The Unix implementation of the Dos unit therefore keeps a table of open directories, and when the table is full, closes one of the directories, and reopens another. This system is adequate but slow if you use a lot of `searchrecs`.

So, to speed up the `findfirst/findnext` system, the `FindClose` call was implemented. When you don't need a `searchrec` any more, you can tell this to the Dos unit by issuing a `FindClose` call. The directory which is kept open for this `searchrec` is then closed, and the table slot freed.

Remark It is recommended to use the Linux call `Glob` when looking for files on Linux.

Errors: Errors are reported in `DosError`.

See also: `FindFirst` (738), `FindNext` (739)

45.10.12 FindFirst

Synopsis: Start search for one or more files.

Declaration: `procedure FindFirst(const path: PathStr; attr: Word; var f: SearchRec)`

Visibility: default

Description: `FindFirst` searches the file specified in `Path`. Normal files, as well as all special files which have the attributes specified in `Attr` will be returned.

It returns a `SearchRec` record for further searching in `F`. `Path` can contain the wildcard characters `?` (matches any single character) and `*` (matches 0 ore more arbitrary characters). In this case `FindFirst` will return the first file which matches the specified criteria. If `DosError` is different from zero, no file(s) matching the criteria was(were) found.

Remark On the EMX target, you cannot issue two different `FindFirst` calls. That is, you must close any previous search operation with `FindClose` (737) before starting a new one. Failure to do so will end in a Run-Time Error 6 (Invalid file handle)

Errors: Errors are reported in `DosError`.

See also: `FindNext` (739), `FindClose` (737)

Listing: `./examples/dosex/ex7.pp`

Program Example7;

uses Dos;

{ Program to demonstrate the FindFirst and FindNext function. }

var

Dir : SearchRec;

begin

FindFirst('*.','archive',Dir);

WriteLn('FileName'+Space(32),'FileSize':9);

while (DosError=0) **do**

begin

WriteLn(Dir.Name+Space(40-Length(Dir.Name)),Dir.Size:9);

FindNext(Dir);

end;

FindClose(Dir);

end.

45.10.13 FindNext

Synopsis: Find next matching file after FindFirst (738).

Declaration: `procedure FindNext (var f: SearchRec)`

Visibility: default

Description: `FindNext` takes as an argument a `SearchRec` from a previous `FindNext` call, or a `FindFirst` call, and tries to find another file which matches the criteria, specified in the `FindFirst` call. If `DosError` is different from zero, no more files matching the criteria were found.

For an example, see `FindFirst` (738).

Errors: `DosError` is used to report errors.

See also: `FindFirst` (738), `FindClose` (737)

45.10.14 FSearch

Synopsis: Search a file in searchpath.

Declaration: `function FSearch (path: PathStr; dirlist: string) : PathStr`

Visibility: default

Description: `FSearch` searches the file `Path` in all directories listed in `DirList`. The full name of the found file is returned. `DirList` must be a list of directories, separated by semi-colons. When no file is found, an empty string is returned.

Remark On Unix systems, `DirList` can also be separated by colons, as is customary on those environments.

Errors: None.

See also: `FExpand` (737)

Listing: `./examples/dosex/ex10.pp`

```

program Example10;

uses Dos;

{ Program to demonstrate the FSearch function. }

var s:pathstr;

begin
  s:=FSearch(ParamStr(1),GetEnv('PATH'));
  if s="" then
    WriteLn(ParamStr(1),' not Found in PATH')
  else
    WriteLn(ParamStr(1),' Found in PATH at ',s);
end.

```

45.10.15 FSplit

Synopsis: Split a full-path filename in parts.

Declaration: `procedure FSplit (path: PathStr; var dir: DirStr; var name: NameStr;
var ext: ExtStr)`

Visibility: default

Description: `FSplit` splits a full file name into 3 parts : A Path, a Name and an extension (in `ext`.) The extension is taken to be all letters after the *last* dot (.). For Dos, however, an exception is made when `LFNSupport=False`, then the extension is defined as all characters after the *first* dot.

Errors: None.

See also: `FSearch` ([739](#))

Listing: `./examples/dosex/ex12.pp`

```

program Example12;

uses Dos;

{ Program to demonstrate the FSplit function. }

var dir:dirstr;
    name:namestr;
    ext:extstr;

begin
  FSplit(ParamStr(1),dir,name,ext);
  WriteLn('Splitted ',ParamStr(1),' in:');
  WriteLn('Path : ',dir);
  WriteLn('Name : ',name);
  WriteLn('Extension: ',ext);
end.

```

45.10.16 GetCBreak

Synopsis: Get control-Break flag.

Declaration: `procedure GetCBreak (var breakvalue: Boolean)`

Visibility: default

Description: `GetCBreak` gets the status of CTRL-Break checking under Dos and Amiga. When `BreakValue` is `false`, then Dos only checks for the CTRL-Break key-press when I/O is performed. When it is set to `True`, then a check is done at every system call.

Remark Under non-Dos and non-Amiga operating systems, `BreakValue` always returns `True`.

Errors: None

See also: `SetCBreak` ([747](#))

45.10.17 GetDate

Synopsis: Get the current date.

Declaration: `procedure GetDate(var year: Word; var month: Word; var mday: Word;
var wday: Word)`

Visibility: default

Description: `GetDate` returns the system's date. `Year` is a number in the range 1980..2099. `mday` is the day of the month, `wday` is the day of the week, starting with Sunday as day 0.

Errors: None.

See also: `GetTime` (745), `SetDate` (748)

Listing: ./examples/dosex/ex2.pp

```

Program Example2;
uses Dos;

{ Program to demonstrate the GetDate function. }

const
  DayStr:array[0..6] of string[3]=('Sun','Mon','Tue','Wed','Thu','Fri','Sat');
  MonthStr:array[1..12] of string[3]=('Jan','Feb','Mar','Apr','May','Jun',
                                     'Jul','Aug','Sep','Oct','Nov','Dec');
var
  Year,Month,Day,WDay : word;
begin
  GetDate(Year,Month,Day,WDay);
  WriteLn('Current date');
  WriteLn(DayStr[WDay],', ',Day,', ',MonthStr[Month],', ',Year,', ');
end.

```

45.10.18 GetEnv

Synopsis: Get environment variable by name.

Declaration: `function GetEnv(envvar: string) : string`

Visibility: default

Description: `Getenv` returns the value of the environment variable `EnvVar`. When there is no environment variable `EnvVar` defined, an empty string is returned.

Remark Under some operating systems (such as Unix), case is important when looking for `EnvVar`.

Errors: None.

See also: `EnvCount` (736), `EnvStr` (736)

Listing: ./examples/dosex/ex14.pp

```

Program Example14;
uses Dos;

{ Program to demonstrate the GetEnv function. }

begin

```

```
WriteLn('Current PATH is ',GetEnv('PATH'));
end.
```

45.10.19 GetFAttr

Synopsis: Get file attributes.

Declaration: `procedure GetFAttr(var f; var attr: Word)`

Visibility: default

Description: `GetFAttr` returns the file attributes of the file-variable `f`. `f` can be a untyped or typed file, or of type `Text`. `f` must have been assigned, but not opened. The attributes can be examined with the following constants :

- `ReadOnly`
- `Hidden`
- `SysFile`
- `VolumeId`
- `Directory`
- `Archive`

Under Linux, supported attributes are:

- `Directory`
- `ReadOnly` if the current process doesn't have access to the file.
- `Hidden` for files whose name starts with a dot (`'.'`).

Errors: Errors are reported in `DosError`

See also: `SetFAttr` ([748](#))

Listing: `./examples/dosex/ex8.pp`

Program Example8;
uses Dos;

{ Program to demonstrate the GetFAttr function. }

```
var
  Attr : Word;
  f : File;
begin
  Assign(f,ParamStr(1));
  GetFAttr(f,Attr);
  WriteLn('File ',ParamStr(1),' has attribute ',Attr);
  if (Attr and archive)<>0 then WriteLn('– Archive');
  if (Attr and directory)<>0 then WriteLn('– Directory');
  if (Attr and readonly)<>0 then WriteLn('– Read–Only');
  if (Attr and sysfile)<>0 then WriteLn('– System');
  if (Attr and hidden)<>0 then WriteLn('– Hidden');
end.
```

45.10.20 GetFTime

Synopsis: Get file last modification time.

Declaration: `procedure GetFTime(var f; var time: LongInt)`

Visibility: default

Description: `GetFTime` returns the modification time of a file. This time is encoded and must be decoded with `UnPackTime`. `F` must be a file type, which has been assigned, and opened.

Errors: Errors are reported in `DosError`

See also: `SetFTime` (749), `PackTime` (747), `UnPackTime` (750)

Listing: `./examples/dosex/ex9.pp`

```

Program Example9;
uses Dos;

{ Program to demonstrate the GetFTime function. }

Function L0(w:word):string;
var
  s : string;
begin
  Str(w,s);
  if w<10 then
    L0:='0'+s
  else
    L0:=s;
end;

var
  f : File;
  Time : Longint;
  DT : DateTime;
begin
  if Paramcount>0 then
    Assign(f,ParamStr(1))
  else
    Assign(f,'ex9.pp' );
  Reset(f);
  GetFTime(f,Time);
  Close(f);
  UnPackTime(Time,DT);
  Write ('File ',ParamStr(1),' is last modified on ');
  Writeln (L0(DT.Month),'-',L0(DT.Day),'-',DT.Year,
           ' at ',L0(DT.Hour),':',L0(DT.Min));
end.

```

45.10.21 GetIntVec

Synopsis: Get interrupt vector.

Declaration: `procedure GetIntVec(intno: Byte; var vector: pointer)`

Visibility: default

Description: `GetIntVec` returns the address of interrupt vector `IntNo`.

Remark This call does nothing, it is present for compatibility only. Modern systems do not allow low level access to the hardware.

Errors: None.

See also: `SetIntVec` (749)

45.10.22 GetLongName

Synopsis: Get the long filename of a DOS 8.3 filename.

Declaration: `function GetLongName (var p: string) : Boolean`

Visibility: default

Description: This function is only implemented in the GO32V2 and Win32 versions of Free Pascal.

`GetLongName` changes the filename `p` to a long filename if the API call to do this is successful. The resulting string is the long file name corresponding to the short filename `p`.

The function returns `True` if the API call was successful, `False` otherwise.

This function should only be necessary when using the DOS extender under Windows 95 and higher.

Errors: If the API call was not successful, `False` is returned.

See also: `GetShortName` (744)

45.10.23 GetMsCount

Synopsis: Number of milliseconds since a starting point.

Declaration: `function GetMsCount : Int64`

Visibility: default

Description: `GetMsCount` returns a number of milliseconds elapsed since a certain moment in time. This moment in time is implementation dependent. This function is used for timing purposes: Subtracting the results of 2 subsequent calls to this function returns the number of milliseconds elapsed between the two calls.

This call is not very reliable, it is recommended to use some system specific calls for timings.

See also: `GetTime` (745)

45.10.24 GetShortName

Synopsis: Get the short (8.3) filename of a long filename.

Declaration: `function GetShortName (var p: string) : Boolean`

Visibility: default

Description: This function is only implemented in the GO32V2 and Win32 versions of Free Pascal.

`GetShortName` changes the filename `p` to a short filename if the API call to do this is successful. The resulting string is the short file name corresponding to the long filename `p`.

The function returns `True` if the API call was successful, `False` otherwise.

This function should only be necessary when using the DOS extender under Windows 95 and higher.

Errors: If the API call was not successful, `False` is returned.

See also: `GetLongName` ([744](#))

45.10.25 `GetTime`

Synopsis: Return the current time.

Declaration: `procedure GetTime(var hour: Word; var minute: Word; var second: Word; var sec100: Word)`

Visibility: default

Description: `GetTime` returns the system's time. `Hour` is on a 24-hour time scale. `sec100` is in hundredth of a second.

Remark Certain operating systems (such as Amiga), always set the `sec100` field to zero.

Errors: None.

See also: `GetDate` ([741](#)), `SetTime` ([749](#))

Listing: `./examples/dosex/ex3.pp`

```

Program Example3;
uses Dos;

{ Program to demonstrate the GetTime function. }

Function L0(w:word):string;
var
  s : string;
begin
  Str(w,s);
  if w<10 then
    L0:='0'+s
  else
    L0:=s;
  end;

var
  Hour,Min,Sec,HSec : word;
begin
  GetTime(Hour,Min,Sec,HSec);
  WriteLn('Current time');
  WriteLn(L0(Hour),':',L0(Min),':',L0(Sec));
end.

```

45.10.26 `GetVerify`

Synopsis: Get verify flag.

Declaration: `procedure GetVerify(var verify: Boolean)`

Visibility: default

Description: `GetVerify` returns the status of the verify flag under Dos. When `Verify` is `True`, then Dos checks data which are written to disk, by reading them after writing. If `Verify` is `False`, then data written to disk are not verified.

Remark Under non-Dos systems (excluding EMX applications running under vanilla DOS), `Verify` is always `True`.

Errors: None.

See also: `SetVerify` ([749](#))

45.10.27 Intr

Synopsis: Execute interrupt.

Declaration: `procedure Intr(intno: Byte; var regs: Registers)`

Visibility: default

Description: `Intr` executes a software interrupt number `IntNo` (must be between 0 and 255), with processor registers set to `Regs`. After the interrupt call returned, the processor registers are saved in `Regs`.

Remark Under non-Dos operating systems, this call does nothing.

Errors: None.

See also: `MSDos` ([746](#))

45.10.28 Keep

Synopsis: Terminate and stay resident.

Declaration: `procedure Keep(exitcode: Word)`

Visibility: default

Description: `Keep` terminates the program, but stays in memory. This is used for TSR (Terminate Stay Resident) programs which catch some interrupt. `ExitCode` is the same parameter as the `Halt` function takes.

Remark This call does nothing, it is present for compatibility only.

Errors: None.

45.10.29 MSDos

Synopsis: Execute MS-DOS system call.

Declaration: `procedure MSDos(var regs: Registers)`

Visibility: default

Description: `MSDos` executes an operating system call. This is the same as doing a `Intr` call with the interrupt number for an OS call.

Remark Under non-Dos operating systems, this call does nothing. On DOS systems, this calls interrupt \$21.

Errors: None.

See also: `Intr` ([746](#))

45.10.30 PackTime

Synopsis: Pack DateTime value to a packed-time format.

Declaration: `procedure PackTime (var t: DateTime; var p: LongInt)`

Visibility: default

Description: `UnPackTime` converts the date and time specified in `T` to a packed-time format which can be fed to `SetFTime`.

Errors: None.

See also: `SetFTime` (749), `FindFirst` (738), `FindNext` (739), `UnPackTime` (750)

Listing: `./examples/dosex/ex4.pp`

```

Program Example4;
uses Dos;

{ Program to demonstrate the PackTime and UnPackTime functions. }

var
  DT : DateTime;
  Time : longint;
begin
  with DT do
    begin
      Year:=2008;
      Month:=11;
      Day:=11;
      Hour:=11;
      Min:=11;
      Sec:=11;
    end;
  PackTime(DT,Time);
  WriteLn('Packed Time : ',Time);
  UnPackTime(Time,DT);
  WriteLn('Unpacked Again:');
  with DT do
    begin
      WriteLn('Year ',Year);
      WriteLn('Month ',Month);
      WriteLn('Day ',Day);
      WriteLn('Hour ',Hour);
      WriteLn('Min ',Min);
      WriteLn('Sec ',Sec);
    end;
end.

```

45.10.31 SetCBreak

Synopsis: Set Control-Break flag status.

Declaration: `procedure SetCBreak (breakvalue: Boolean)`

Visibility: default

Description: `SetCBreak` sets the status of CTRL-Break checking. When `BreakValue` is `false`, then `Dos` only checks for the CTRL-Break key-press when I/O is performed. When it is set to `True`, then a check is done at every system call.

Remark Under non-Dos and non-Amiga operating systems, this call does nothing.

Errors: None.

See also: `GetCBreak` (740)

45.10.32 SetDate

Synopsis: Set system date.

Declaration: `procedure SetDate(year: Word; month: Word; day: Word)`

Visibility: default

Description: `SetDate` sets the system's internal date. `Year` is a number between 1980 and 2099.

Remark On a Unix machine, there must be root privileges, otherwise this routine will do nothing. On other Unix systems, this call currently does nothing.

Errors: None.

See also: `GetDate` (741), `SetTime` (749)

45.10.33 SetFAttr

Synopsis: Set file attributes.

Declaration: `procedure SetFAttr(var f; attr: Word)`

Visibility: default

Description: `SetFAttr` sets the file attributes of the file-variable `F`. `F` can be a untyped or typed file, or of type `Text`. `F` must have been assigned, but not opened. The attributes can be a sum of the following constants:

- `ReadOnly`
- `Hidden`
- `SysFile`
- `VolumeId`
- `Directory`
- `Archive`

Remark Under Unix like systems (such as Linux and BeOS) the call exists, but is not implemented, i.e. it does nothing.

Errors: Errors are reported in `DosError`.

See also: `GetFAttr` (742)

45.10.34 SetFTime

Synopsis: Set file modification time.

Declaration: `procedure SetFTime(var f; time: LongInt)`

Visibility: default

Description: `SetFTime` sets the modification time of a file, this time is encoded and must be encoded with `PackTime`. `F` must be a file type, which has been assigned, and opened.

Remark Under Unix like systems (such as Linux and BeOS) the call exists, but is not implemented, i.e. it does nothing.

Errors: Errors are reported in `DosError`

See also: `GetFTime` (743), `PackTime` (747), `UnPackTime` (750)

45.10.35 SetIntVec

Synopsis: Set interrupt vector.

Declaration: `procedure SetIntVec(intno: Byte; vector: pointer)`

Visibility: default

Description: `SetIntVec` sets interrupt vector `IntNo` to `Vector`. `Vector` should point to an interrupt procedure.

Remark This call does nothing, it is present for compatibility only.

Errors: None.

See also: `GetIntVec` (743)

45.10.36 SetTime

Synopsis: Set system time.

Declaration: `procedure SetTime(hour: Word; minute: Word; second: Word; sec100: Word)`

Visibility: default

Description: `SetTime` sets the system's internal clock. The `Hour` parameter is on a 24-hour time scale.

Remark On a Linux machine, there must be root privileges, otherwise this routine will do nothing. On other Unix systems, this call currently does nothing.

Errors: None.

See also: `GetTime` (745), `SetDate` (748)

45.10.37 SetVerify

Synopsis: Set verify flag.

Declaration: `procedure SetVerify(verify: Boolean)`

Visibility: default

Description: `SetVerify` sets the status of the verify flag under Dos. When `Verify` is `True`, then Dos checks data which are written to disk, by reading them after writing. If `Verify` is `False`, then data written to disk are not verified.

Remark Under non-Dos operating systems (excluding EMX applications running under vanilla Dos), `Verify` is always `True`.

Errors: None.

See also: `SetVerify` ([749](#))

45.10.38 SwapVectors

Synopsis: Swap interrupt vectors.

Declaration: `procedure SwapVectors`

Visibility: `default`

Description: `SwapVectors` swaps the contents of the internal table of interrupt vectors with the current contents of the interrupt vectors. This is called typically in before and after an `Exec` call.

Remark Under certain operating systems, this routine may be implemented as an empty stub.

Errors: None.

See also: `Exec` ([737](#)), `SetIntVec` ([749](#))

45.10.39 UnixDateToDt

Synopsis: Convert a Unix timestamp to a `DateTime` record.

Declaration: `procedure UnixDateToDt (SecsPast: LongInt; var Dt: DateTime)`

Visibility: `default`

Description: `DTToUnixDate` converts the Unix timestamp value in `SecsPast` to a `DateTime` representation in `DT`. It is an internal function, implemented on Unix platforms, and should not be used.

Errors: None.

See also: `DTToUnixDate` ([735](#)), `PackTime` ([747](#)), `UnpackTime` ([750](#)), `GetTime` ([745](#)), `SetTime` ([749](#))

45.10.40 UnpackTime

Synopsis: Unpack packed file time to a `DateTime` value.

Declaration: `procedure UnpackTime (p: LongInt; var t: DateTime)`

Visibility: `default`

Description: `UnPackTime` converts the file-modification time in `p` to a `DateTime` record. The file-modification time can be returned by `GetFTime`, `FindFirst` or `FindNext` calls.

For an example, see `PackTime` ([747](#)).

Errors: None.

See also: `GetFTime` ([743](#)), `FindFirst` ([738](#)), `FindNext` ([739](#)), `PackTime` ([747](#))

45.10.41 weekday

Synopsis: Return the day of the week.

Declaration: `function weekday(y: LongInt; m: LongInt; d: LongInt) : LongInt`

Visibility: default

Description: `WeekDay` returns the day of the week on which the day Y/M/D falls. Sunday is represented by 0, Saturday is 6.

Errors: On error, -1 is returned.

See also: `PackTime` (747), `UnpackTime` (750), `GetTime` (745), `SetTime` (749)

45.11 DateTime

```

DateTime = packed record
  Year : Word;
  Month : Word;
  Day : Word
;
  Hour : Word;
  Min : Word;
  Sec : Word;
end

```

The `DateTime` type is used in `PackTime` (747) and `UnPackTime` (750) for setting/reading file times with `GetFTime` (743) and `SetFTime` (749).

45.12 SearchRec

```

SearchRec = packed record
  SearchPos : TOff;
  SearchNum : LongInt
;
  DirPtr : Pointer;
  SearchType : Byte;
  SearchAttr : Byte;
  Mode : Word;
  Fill : Array[1..1] of Byte;
  Attr : Byte;
  Time
  : LongInt;
  Size : LongInt;
  Reserved : Word;
  Name : string;
  SearchSpec : string;
  NamePos : Word;
end

```

`SearchRec` is filled by the `FindFirst` (738) call and can be used in subsequent `FindNext` (739) calls to search for files. The structure of this record depends on the platform. Only the following fields are present on all platforms:

Attr File attributes.

Time File modification time.

Size File size

Name File name (name part only, no path)

Mode File access mode (Linux only)

Chapter 46

Reference for unit 'dxeload'

46.1 Used units

Table 46.1: Used units by unit 'dxeload'

Name	Page
System	1335

46.2 Overview

The `dxeload` unit was implemented by Pierre Mueller for dos, it allows to load a DXE file (an object file with 1 entry point) into memory and return a pointer to the entry point.

It exists only for dos.

46.3 Procedures and functions

46.3.1 `dxeload`

Synopsis: Load DXE file in memory.

Declaration: `function dxeload(filename: string) : pointer`

Visibility: default

Description: `dxeload` loads the contents of the file `filename` into memory. It performs the necessary relocations in the object code, and returns then a pointer to the entry point of the code.

For an example, see the `emu387` ([758](#)) unit in the RTL.

Errors: If an error occurs during the load or relocations, `Nil` is returned.

Chapter 47

Reference for unit 'dynlibs'

47.1 Used units

Table 47.1: Used units by unit 'dynlibs'

Name	Page
System	1335

47.2 Overview

The Dynlibs unit provides support for dynamically loading shared libraries. It is available only on those platforms that support shared libraries. The functionality available here may only be a part of the functionality available on each separate platform, in the interest of portability.

On UNIX platforms, using this unit will cause the program to be linked to the C library, as most shared libraries are implemented in C and the dynamical linker too.

47.3 Constants, types and variables

47.3.1 Constants

`NilHandle = System.NilHandle`

Correctly typed Nil handle - returned on error by `LoadLibrary` ([756](#)).

`SharedSuffix = System.SharedSuffix`

`SharedSuffix` contains the extension of a shared library (dynamically loadable library) on the current platform. It does not contain the . (dot) character. This can be used to determine the name of a shared library in a platform independent way.

47.3.2 Types

`HModule = TLibHandle`

Alias for `TLibHandle` (755) type.

```
TLibHandle = System.TLibHandle
```

`TLibHandle` should be considered an opaque type. It is defined differently on various platforms. The definition shown here depends on the platform for which the documentation was generated.

47.4 Procedures and functions

47.4.1 FreeLibrary

Synopsis: For compatibility with Delphi/Windows: Unload a library.

Declaration: `function FreeLibrary(Lib: TLibHandle) : Boolean`

Visibility: default

Description: `FreeLibrary` provides the same functionality as `UnloadLibrary` (757), and is provided for compatibility with Delphi.

See also: `UnloadLibrary` (757)

47.4.2 GetLoadErrorStr

Synopsis: Return a description of the last error during load/get procedure address operations.

Declaration: `function GetLoadErrorStr : string`

Visibility: default

Description: `GetLoadErrorStr` returns a textual description of the last library loading or unloading error, or a call to `GetProcAddress` (756). No other system calls may be made between the load call and the call of `GetLoadErrorStr`.

See also: `LoadLibrary` (756), `GetProcAddress` (756)

47.4.3 GetProcAddress

Synopsis: For compatibility with Delphi/Windows: Get the address of a procedure.

Declaration: `function GetProcAddress(Lib: TLibHandle; const ProcName: AnsiString)
: Pointer`

Visibility: default

Description: `GetProcAddress` provides the same functionality as `GetProcAddress` (756), and is provided for compatibility with Delphi.

See also: `GetProcAddress` (756)

47.4.4 GetProcedureAddress

Synopsis: Get the address of a procedure or symbol in a dynamic library.

Declaration:

```
function GetProcedureAddress(Lib: TLibHandle;
                             const ProcName: AnsiString) : Pointer
function GetProcedureAddress(Lib: TLibHandle; Ordinal: TOrdinalEntry)
                             : Pointer
```

Visibility: default

Description: `GetProcedureAddress` returns a pointer to the location in memory of the symbol `ProcName` or ordinal value `Ordinal` in the dynamically loaded library specified by its handle `lib`. If the symbol cannot be found or the handle is invalid, `Nil` is returned.

On Windows, only an exported procedure or function can be searched this way. On Unix platforms the location of any exported symbol can be retrieved this way.

Only windows and OS/2 support getting the address of a function using an ordinal value.

Errors: If the symbol cannot be found, `Nil` is returned.

See also: [LoadLibrary \(756\)](#), [UnloadLibrary \(757\)](#)

47.4.5 LoadLibrary

Synopsis: Load a dynamic library and return a handle to it.

Declaration:

```
function LoadLibrary(const Name: RawByteString) : TLibHandle
function LoadLibrary(const Name: UnicodeString) : TLibHandle
```

Visibility: default

Description: `LoadLibrary` loads a dynamic library in file `Name` and returns a handle to it. If the library cannot be loaded, `NilHandle (754)` is returned.

No assumptions should be made about the location of the loaded library if a relative pathname is specified. The behaviour is dependent on the platform. Therefore it is best to specify an absolute pathname if possible.

Errors: On error, `NilHandle (754)` is returned.

See also: [UnloadLibrary \(757\)](#), [GetProcedureAddress \(756\)](#)

47.4.6 SafeLoadLibrary

Synopsis: Saves the control word and loads a library.

Declaration:

```
function SafeLoadLibrary(const Name: RawByteString) : TLibHandle
function SafeLoadLibrary(const Name: UnicodeString) : TLibHandle
```

Visibility: default

Description: `SafeLoadLibrary` saves the FPU control word, and calls `LoadLibrary (756)` with library name `Name`. After that function has returned, the FPU control word is saved again. (only on Intel i386 CPUs).

See also: [LoadLibrary \(756\)](#)

47.4.7 UnloadLibrary

Synopsis: Unload a previously loaded library.

Declaration: `function UnloadLibrary(Lib: TLibHandle) : Boolean`

Visibility: `default`

Description: `UnloadLibrary` unloads a previously loaded library (specified by the handle `lib`). The call returns `True` if successful, `False` otherwise.

Errors: On error, `False` is returned.

See also: `LoadLibrary` ([756](#)), `GetProcAddress` ([756](#))

Chapter 48

Reference for unit 'emu387'

48.1 Used units

Table 48.1: Used units by unit 'emu387'

Name	Page
System	1335

48.2 Overview

The `emu387` unit was written by Pierre Mueller for dos. It sets up the coprocessor emulation for FPC under dos. It is not necessary to use this unit on other OS platforms because they either simply do not run on a machine without coprocessor, or they provide the coprocessor emulation themselves.

It shouldn't be necessary to use the function in this unit, it should be enough to place this unit in the `uses` clause of your program to enable the coprocessor emulation under dos. The unit initialization code will try and load the coprocessor emulation code and initialize it.

48.3 Procedures and functions

48.3.1 `npxsetup`

Synopsis: Set up coprocessor emulation.

Declaration: `procedure npxsetup(prog_name: string)`

Visibility: `default`

Description: `npxsetup` checks whether a coprocessor is found. If not, it loads the file `wmemu387.dxe` into memory and initializes the code in it.

If the environment variable `387` is set to `N`, then the emulation will be loaded, even if there is a coprocessor present. If the variable doesn't exist, or is set to any other value, the unit will try to detect the presence of a coprocessor unit.

The function searches the file `wmemu387.dxe` in the following way:

- 1.If the environment variable `EMU387` is set, then it is assumed to point at the `wmemu387.dxe` file.
- 2.if the environment variable `EMU387` does not exist, then the function will take the path part of `prog_name` and look in that directory for the file `wmemu387.dxe`.

It should never be necessary to call this function, because the initialization code of the unit contains a call to the function with as an argument `paramstr(0)`. This means that you should deliver the file `wmemu387.dxe` together with your program.

Errors: If there is an error, an error message is printed to standard error, and the program is halted, since any floating-point code is bound to fail anyhow.

Chapter 49

Reference for unit 'errors'

49.1 Used units

Table 49.1: Used units by unit 'errors'

Name	Page
System	1335
unixtype	2229

49.2 Overview

The errors unit contains routines to convert a UNIX system call error code to an error message: `StrError` ([762](#)). It is only available on UNIX platforms.

49.3 Constants, types and variables

49.3.1 Constants

```
sys_errlist : Array[0..sys_errn-1] of PChar = ('Success', 'Operation not permitted',
, 'No such file or directory', 'No such process', 'Interrupted system call'
, 'I/O error', 'No such device or address', 'Arg list too long', 'Exec format erro
, 'Bad file number', 'No child processes', 'Try again', 'Out of memory'
, 'Permission denied', 'Bad address', 'Block device required', 'Device or resource
, 'File exists', 'Cross-device link', 'No such device', 'Not a directory'
, 'Is a directory', 'Invalid argument', 'File table overflow', 'Too many open file
, 'Not a typewriter', 'Text (code segment) file busy', 'File too large'
, 'No space left on device', 'Illegal seek', 'Read-only file system'
, 'Too many links', 'Broken pipe', 'Math argument out of domain of func'
, 'Math result not representable', 'Resource deadlock would occur'
, 'File name too long', 'No record locks available', 'Function not implemented'
, 'Directory not empty', 'Too many symbolic links encountered', 'Operation would b
, 'No message of desired type', 'Identifier removed', 'Channel number out of range
, 'Level 2 not synchronized', 'Level 3 halted', 'Level 3 reset', 'Link number out
, 'Protocol driver not attached', 'No CSI structure available', 'Level 2 halted'
```

```
, 'Invalid exchange', 'Invalid request descriptor', 'Exchange full'
, 'No anode', 'Invalid request code', 'Invalid slot', 'File locking deadlock error'
, 'Bad font file format', 'Device not a stream', 'No data available'
, 'Timer expired', 'Out of streams resources', 'Machine is not on the network'
, 'Package not installed', 'Object is remote', 'Link has been severed'
, 'Advertise error', 'Srmount error', 'Communication error on send'
, 'Protocol error', 'Multihop attempted', 'RFS specific error', 'Not a data message'
, 'Value too large for defined data type', 'Name not unique on network'
, 'File descriptor in bad state', 'Remote address changed', 'Can not access a needed file'
, 'Accessing a corrupted shared library', '.lib section in a.out corrupted'
, 'Attempting to link in too many shared libraries', 'Cannot exec a shared library'
, 'Illegal byte sequence', 'Interrupted system call should be restarted'
, 'Streams pipe error', 'Too many users', 'Socket operation on non-socket'
, 'Destination address required', 'Message too long', 'Protocol wrong type for socket'
, 'Protocol not available', 'Protocol not supported', 'Socket type not supported'
, 'Operation not supported on transport endpoint', 'Protocol family not supported'
, 'Address family not supported by protocol', 'Address already in use'
, 'Cannot assign requested address', 'Network is down', 'Network is unreachable'
, 'Network dropped connection because of reset', 'Software caused connection abort'
, 'Connection reset by peer', 'No buffer space available', 'Transport endpoint is already connected'
, 'Transport endpoint is not connected', 'Cannot send after transport endpoint shutdown'
, 'Too many references: cannot splice', 'Connection timed out', 'Connection refused'
, 'Host is down', 'No route to host', 'Operation already in progress'
, 'Operation now in progress', 'Stale NFS file handle', 'Structure needs cleaning'
, 'Not a XENIX named type file', 'No XENIX semaphores available',
'Is a named type file', 'Remote I/O error', 'Quota exceeded', 'No medium found'
, 'Wrong medium type')
```

`sys_errn` is an array with then error codes for the current operating system. It should not be used directly, instead use `StrError` (762).

```
sys_errn = 125
```

`sys_errn` is the number of error codes for the current operating system. It should not be used directly, instead use `StrError` (762).

49.4 Procedures and functions

49.4.1 PError

Synopsis: Print error on standard error output.

Declaration: `procedure PError(const s: string; Errno: cint)`

Visibility: default

Description: `PError` will print the error message `S` followed by the `errNo` and the result for `StrError(errNo)` for `ErrNo` on standard output.

Errors: None.

See also: `StrError` (762)

49.4.2 StrError

Synopsis: Convert an error code to a string.

Declaration: `function StrError(err: cint) : string`

Visibility: default

Description: `StrError` will convert the error code `err` to a string.

Errors: If the error code is unknown or out of bounds, an 'Unknown error (err)' string will be returned.

See also: `PError` ([761](#))

Chapter 50

Reference for unit 'exeinfo'

50.1 Used units

Table 50.1: Used units by unit 'exeinfo'

Name	Page
System	1335

50.2 Overview

The `exeinfo` unit implements some cross-platform routines to examine the contents of an executable: information about sections, mapping addresses to loaded modules etc.

It is mainly used by the `lineinfo` ([994](#)) and `Infodwrf` ([1027](#)) unit to examine the binary for debug info.

50.3 Constants, types and variables

50.3.1 Types

```
TExeOffset = PtrUInt
```

The exact type of `TExeOffset` will differ per platform.

```
TExeProcessAddress = PtrUInt
```

The exact type of `TExeProcessAddress` will differ per platform.

50.4 Procedures and functions

50.4.1 CloseExeFile

Synopsis: Close a previously opened file.

Declaration: `function CloseExeFile(var e: TExeFile) : Boolean`

Visibility: default

Description: `CloseExeFile` closes an executable file image previously opened with `OpenExeFile` (764), and represented by `e`.

The function returns `True` if the file was closed successfully, or `False` if something went wrong.

Errors: In case of an error, `False` is returned.

See also: `OpenExeFile` (764)

50.4.2 FindExeSection

Synopsis: Find a section in the binary image.

Declaration:

```
function FindExeSection(var e: TExeFile; const secname: string;
                        var secofs: LongInt; var seclen: LongInt)
                        : Boolean
```

Visibility: default

Description: `FindExeSection` examines the binary that was opened with `OpenExeFile` (764) (represented by `e`) and searches for the section named `secname`. If found, the section offset is returned in `secofs` and the section length (in bytes) is returned in `seclen`.

The function returns `True` if the section was found, `False` if not.

See also: `OpenExeFile` (764)

50.4.3 GetModuleByAddr

Synopsis: Return the module name by address.

Declaration:

```
procedure GetModuleByAddr(addr: pointer; var baseaddr: pointer;
                          var filename: string)
```

Visibility: default

Description: `GetModuleByAddr` returns the name of the module that contains address `addr`. If successful, it returns `True` and returns the filename in `FileName` and the base address at which it is loaded in `BaseAddr`.

50.4.4 OpenExeFile

Synopsis: Open an executable file.

Declaration:

```
function OpenExeFile(var e: TExeFile; const fn: string) : Boolean
```

Visibility: default

Description: `OpenExeFile` opens the executable file `fn` and initializes the structure `e` for subsequent calls to routines in the `exeinfo` unit.

The function returns `True` if the file was opened successfully, false otherwise.

See also: `FindExeSection` (764), `CloseExeFile` (763), `ReadDebugLink` (765)

50.4.5 ReadDebugLink

Synopsis: Read the location of a debug info filename.

Declaration: `function ReadDebugLink(var e: TExeFile; var dbgfn: string) : Boolean`

Visibility: default

Description: `ReadDebugLink` examines the `.gnu_debuglink` section to see if the debug information is stored in an external file. If so, then the name of the file with the debug information is returned in the `dbgfn` parameter.

The function returns `false` if there is no external debug information file, or if the file with debug information does not exist. It is searched next to the binary file or in the current directory.

See also: `OpenExeFile` ([764](#)), `CloseExeFile` ([763](#))

50.5 TExeFile

```
TExeFile = record
  f : File;
  size : Int64;
  isopen : Boolean;
  nsects : LongInt;
  sechdrofs : TExeOffset;
  secstrofs : TExeOffset
;
  processaddress : TExeProcessAddress;
  FunctionRelative : Boolean
;
  ImgOffset : TExeOffset;
  filename : string;
  buf : Array[0..
    .4095] of Byte;
  bufsize : LongInt;
  bufcnt : LongInt;
end
```

`TExeFile` is a record used in the various calls of this unit. It contains a file descriptor, and various fields that describe the executable.

The structure of `TExeFile` is opaque, that is, one shouldn't rely on the exactness of this structure, it may change any time in the future.

Chapter 51

Reference for unit 'fgl'

51.1 Used units

Table 51.1: Used units by unit 'fgl'

Name	Page
System	1335
sysutils	1609
Types	2007

51.2 Overview

The `fgl` unit contains some basic list-related generic classes.

51.3 Constants, types and variables

51.3.1 Constants

`MaxGListSize = MaxInt div 1024`~~deprecated~~

`MaxGListSize` is the maximum number of elements in the `TFPGList` ([771](#)) list.

`MaxListSize = Maxint div 16`

`MaxListSize` is the maximum number of elements a list can contain before the memory runs out.

51.3.2 Types

`TFPSListCompareFunc = function(Key1: Pointer; Key2: Pointer) : Integer
of object`

`TFPSListCompareFunc` is used in the `TFPSList.Sort` ([803](#)) method to compare 2 elements. The list passes 2 pointers to the actual items to the compare function. The result of this function determines how the pointers will be sorted:

- If the result of this function is negative, the first key (`key1`) is assumed to be 'less' than the second key (`key2`) and will be moved before the second in the list.
- If the function result is positive, the first key (`key1`) pointer is assumed to be 'greater than' the second key (`key2`) and will be moved after the second in the list.
- if the function result is zero, the keys are assumed to be 'equal' and no moving will take place.

51.4 EListError

51.4.1 Description

`EListError` is the exception used in the `TFPSList` (798) class to indicate errors such as a list index out of bounds, wrong capacity etc.

See also: `TFPSList.Capacity` (804), `TFPSList.Exchange` (801), `TFPSList.Items` (804)

51.5 TFPGInterfacedObjectList

51.5.1 Description

`TFPGList` can be used to specialize a list for any class type `T` that requires reference counting (all objects that implement `IInterface` or `IUnknown`). It will specialize to a list with the same methods as `TFPSList` (798) or classes `TFPList` (766) or `TFPObjectList`

Classes that implement `IInterface` or `IUnknown` require special care to maintain the reference count. The `TFPGInterfacedObjectList` list provides the necessary functionality to deal with this.

See also: `TFPSList` (798), classes `TFPList` (766)

51.5.2 Method overview

Page	Method	Description
768	<code>Add</code>	Add new object of class <code>T</code> to the list.
769	<code>AddList</code>	Adds the elements from another list.
769	<code>Assign</code>	Copy objects from Source list.
768	<code>Create</code>	Instantiate a new interfaced object list.
768	<code>Extract</code>	Extract an item from the list.
768	<code>GetEnumerator</code>	Return a list enumerator for <code>T</code> .
769	<code>IndexOf</code>	Index of object.
769	<code>Insert</code>	Insert a new object in the list.
770	<code>Remove</code>	Remove an object from the list.
770	<code>Sort</code>	Sort the objects in the list.

51.5.3 Property overview

Page	Properties	Access	Description
770	<code>First</code>	rw	First non-nil object.
771	<code>Items</code>	rw	Indexed access to objects in the list.
771	<code>Last</code>	rw	Last non- <code>Nil</code> object.
771	<code>List</code>	r	Internal list pointer.

51.5.4 TFPGInterfacedObjectList.Create

Synopsis: Instantiate a new interfaced object list.

Declaration: `constructor Create`

Visibility: `public`

Description: `Create` instantiates a new object list. It will simply call the inherited constructor with the correct item size.

See also: `TFPSList.Destroy` ([799](#))

51.5.5 TFPGInterfacedObjectList.Add

Synopsis: Add new object of class `T` to the list.

Declaration: `function Add(const Item: T) : Integer`

Visibility: `public`

Description: `Add` adds a new item `Item` of class type `T` to the list and returns the position at which the item was added. `Add` will increase the reference count of the object.

Errors: If the item could not be added, an `EListError` ([767](#)) exception is raised.

See also: `TFPGInterfacedObjectList.Extract` ([768](#)), `TFPGInterfacedObjectList.Items` ([771](#)), `TFPGInterfacedObjectList.IndexOf` ([769](#))

51.5.6 TFPGInterfacedObjectList.Extract

Synopsis: Extract an item from the list.

Declaration: `function Extract(const Item: T) : T`

Visibility: `public`

Description: `Extract` removes `Item` from the list and returns the removed item, or `Nil` if it was not found. The extracted object will not be destroyed.

Errors: None.

See also: `TFPSList.Delete` ([800](#))

51.5.7 TFPGInterfacedObjectList.GetEnumerator

Synopsis: Return a list enumerator for `T`.

Declaration: `function GetEnumerator : TFPGListEnumeratorSpec`

Visibility: `public`

Description: `GetEnumerator` returns an enumerator for the elements in the list. It is a specialized version of `TFPGListEnumerator` ([776](#)).

See also: `TFPGListEnumerator` ([776](#))

51.5.8 TFPGInterfacedObjectList.IndexOf

Synopsis: Index of object.

Declaration: `function IndexOf(const Item: T) : Integer`

Visibility: public

Description: `IndexOf` returns the index of `Item` in the list, or -1 if the object does not appear in the list.

Errors: None.

See also: `TFPGInterfacedObjectList.Items` (771), `TFPGInterfacedObjectList.Insert` (769), `TFPGInterfacedObjectList.Add` (768)

51.5.9 TFPGInterfacedObjectList.Insert

Synopsis: Insert a new object in the list.

Declaration: `procedure Insert(Index: Integer; const Item: T)`

Visibility: public

Description: `Insert` inserts a new object (`Item`) in the list at position `Index`. The index is zero based and must be less than `Count` (804).

Errors: If an invalid index is specified, an `EListError` (767) exception is raised.

See also: `TFPGInterfacedObjectList.Items` (771), `TFPGInterfacedObjectList.Add` (768)

51.5.10 TFPGInterfacedObjectList.Assign

Synopsis: Copy objects from Source list.

Declaration: `procedure Assign(Source: TFPGInterfacedObjectList)`

Visibility: public

Description: `Assign` clears the list and copies all items in `Source` to the list. The source list must be of the same type as the destination list.

See also: `TFPSList.Clear` (800), `TFPGObjectList.Add` (794)

51.5.11 TFPGInterfacedObjectList.AddList

Synopsis: Adds the elements from another list.

Declaration: `procedure AddList(Source: TFPGInterfacedObjectList)`

Visibility: public

Description: `AddList` adds all the elements from list `Source` to the current list.

See also: `TFPGList.AddList` (774), `TFPSList.AddList` (803), `TFPGObjectList.AddList` (795), `TFPGInterfacedObjectList.Add` (768)

51.5.12 TFPGInterfacedObjectList.Remove

Synopsis: Remove an object from the list.

Declaration: `function Remove(const Item: T) : Integer`

Visibility: public

Description: `Remove` removes the object `Item` from the list, and returns the index of the removed item. If no item was removed, `-1` is returned. Only the first object is removed.

Removing an object from the list may cause the object to be freed.

Errors: None.

See also: `TFPGInterfacedObjectList.IndexOf` (769), `TFPSList.Delete` (800), `TFPGInterfacedObjectList.FreeObjects` (767)

51.5.13 TFPGInterfacedObjectList.Sort

Synopsis: Sort the objects in the list.

Declaration: `procedure Sort(Compare: TCompareFunc)`

Visibility: public

Description: `Sort` sorts the elements in the list using the provided `Compare` function. The list passes 2 items to the compare function. The result of this function determines how the items will be sorted:

- If the result of this function is negative, the first object (`Item1`) is assumed to be 'less' than the second object (`Item2`) and will be moved before the second in the list.
- If the function result is positive, the first object (`Item1`) is assumed to be 'greater than' the second object (`Item2`) and will be moved after the second in the list.
- if the function result is zero, the objects are assumed to be 'equal' and no moving will take place.

Errors: None.

See also: `TFPSList.Sorted` (798)

51.5.14 TFPGInterfacedObjectList.First

Synopsis: First non-nil object.

Declaration: `Property First : T`

Visibility: public

Access: Read, Write

Description: `First` returns the first non-nil object. If no such element is present, `Nil` is returned.

See also: `TFPSList.First` (805), `TFPGInterfacedObjectList.Last` (771), `TFPSList.Pack` (803)

51.5.15 TFPGInterfacedObjectList.Last

Synopsis: Last non-`Nil` object.

Declaration: `Property Last : T`

Visibility: `public`

Access: `Read, Write`

Description: `Last` returns the last non-`Nil` object. If no such element is present, `Nil` is returned.

See also: `TFPGInterfacedObjectList.First` (770), `TFPSList.Last` (805)

51.5.16 TFPGInterfacedObjectList.Items

Synopsis: Indexed access to objects in the list.

Declaration: `Property Items[Index: Integer]: T; default`

Visibility: `public`

Access: `Read, Write`

Description: `Items` provides indexed access to the objects in the list. The objects can be get or set.

The index `Index` is zero based, and has a maximum value of `Count-1` (804).

The previous object at position `Index` may be freed when setting the property, depending on its reference count.

Errors: If an invalid index is used, an `EListError` (767) exception is raised.

See also: `TFPSList.Count` (804)

51.5.17 TFPGInterfacedObjectList.List

Synopsis: Internal list pointer.

Declaration: `Property List : PTypeList`

Visibility: `public`

Access: `Read`

Description: `List` is the internal list of objects. It should not be used directly.

See also: `TFPGInterfacedObjectList.Items` (771)

51.6 TFPGList

51.6.1 Description

`TFPGList` can be used to specialize a list for any type `T` that does not require reference counting (such as interfaced objects). It will specialize to a list with the same methods as `TFPSList` (798) or `classes.TFPList` (766)

See also: `TFPSList` (798), `classes.TFPList` (766)

51.6.2 Method overview

Page	Method	Description
772	Add	Add new item of type T to the list.
774	AddList	Adds the elements from another list.
774	Assign	Copy elements from Source list.
772	Create	Instantiate a new list.
773	Extract	Extract an item from the list.
773	GetEnumerator	Return a list enumerator for T.
773	IndexOf	Index of item.
773	Insert	Insert a new item in the list.
772	ItemsManaged	Indicates whether the list item is a managed type or not.
774	Remove	Remove an item from the list.
774	Sort	Sort the list.

51.6.3 Property overview

Page	Properties	Access	Description
775	First	rw	First non-empty item.
775	Items	rw	Indexed access to items in the list.
775	Last	rw	Last non-empty item.
775	List	r	Internal list object.

51.6.4 TFPGList.Create

Synopsis: Instantiate a new list.

Declaration: `constructor Create`

Visibility: `public`

Description: `Create` instantiates a new list. It will simply call the inherited constructor with the correct item size: `sizeof(T)`.

51.6.5 TFPGList.ItemsManaged

Synopsis: Indicates whether the list item is a managed type or not.

Declaration: `class function ItemIsManaged : Boolean; Override`

Visibility: `public`

Description: `ItemIsManaged` indicates whether the list item type is a managed type or not. If not, faster codepaths are sometimes taken in the code.

51.6.6 TFPGList.Add

Synopsis: Add new item of type T to the list.

Declaration: `function Add(const Item: T) : Integer`

Visibility: `public`

Description: `Add` adds a new item `Item` of generic type T to the list and returns the position at which the item was added.

Errors: If the item could not be added, an `EListError` (767) exception is raised.

See also: `TFPGList.Extract` (773), `TFPGList.Items` (775), `TFPGList.IndexOf` (773)

51.6.7 TFPGList.Extract

Synopsis: Extract an item from the list.

Declaration: `function Extract(const Item: T) : T`

Visibility: public

Description: `Extract` removes `Item` from the list and returns the removed item, or an expression equivalent to `T(0)` if it was not found.

Errors: None.

See also: `TFPSList.Delete` (800)

51.6.8 TFPGList.GetEnumerator

Synopsis: Return a list enumerator for T.

Declaration: `function GetEnumerator : TFPGListEnumeratorSpec`

Visibility: public

Description: `GetEnumerator` returns an enumerator for the elements in the list. It is a specialized version of `TFPGListEnumerator` (776).

See also: `TFPGListEnumerator` (776)

51.6.9 TFPGList.IndexOf

Synopsis: Index of item.

Declaration: `function IndexOf(const Item: T) : Integer`

Visibility: public

Description: `IndexOf` returns the index of `Item` in the list, or -1 if the item does not appear in the list.

Errors: None.

See also: `TFPGList.Items` (775), `TFPGList.Insert` (773), `TFPSList.Add` (799)

51.6.10 TFPGList.Insert

Synopsis: Insert a new item in the list.

Declaration: `procedure Insert(Index: Integer; const Item: T)`

Visibility: public

Description: `Insert` inserts a new item in the list at position `Index`. The index is zero based and must be less than `Count` (804).

Errors: If an invalid index is specified, an `EListError` (767) exception is raised.

See also: `TFPGList.Items` (775), `TFPGList.Insert` (773), `TFPSList.Add` (799)

51.6.11 TFPGList.Assign

Synopsis: Copy elements from Source list.

Declaration: `procedure Assign(Source: TFPGList)`

Visibility: public

Description: `Assign` clears the list and copies all items in `Source` to the list. The source list must be of the same type as the destination list.

See also: `TFPSList.Clear` (800), `TFPGList.Add` (772)

51.6.12 TFPGList.AddList

Synopsis: Adds the elements from another list.

Declaration: `procedure AddList(Source: TFPGList)`

Visibility: public

Description: `AddList` adds all the elements from list `Source` to the current list.

See also: `TFPGList.Add` (772), `TFPSList.AddList` (803), `TFPGObjectList.AddList` (795)

51.6.13 TFPGList.Remove

Synopsis: Remove an item from the list.

Declaration: `function Remove(const Item: T) : Integer`

Visibility: public

Description: `Remove` removes the item `Item` from the list, and returns the index of the removed item. If no item was removed, -1 is returned. Only the first item is removed.

Errors: None.

See also: `TFPGList.IndexOf` (773)

51.6.14 TFPGList.Sort

Synopsis: Sort the list.

Declaration: `procedure Sort(Compare: TCompareFunc)`

Visibility: public

Description: `Sort` sorts the elements in the list using the provided `Compare` function. The list passes 2 items to the compare function. The result of this function determines how the items will be sorted:

- If the result of this function is negative, the first item (`Item1`) is assumed to be 'less' than the second item (`Item2`) and will be moved before the second in the list.
- If the function result is positive, the first item (`Item1`) is assumed to be 'greater than' the second item (`Item2`) and will be moved after the second in the list.
- if the function result is zero, the items are assumed to be 'equal' and no moving will take place.

51.6.15 TFPGList.First

Synopsis: First non-empty item.

Declaration: `Property First : T`

Visibility: public

Access: Read,Write

Description: `First` returns the first non-empty item, which means the first item not equal to `T ()`. If no such element is present, `T ()` is returned.

See also: `TFPSList.First` (805), `TFPGList.Last` (775)

51.6.16 TFPGList.Last

Synopsis: Last non-empty item.

Declaration: `Property Last : T`

Visibility: public

Access: Read,Write

Description: `Last` returns the last non-empty item, which means the last item not equal to `T ()`. If no such element is present, `T ()` is returned.

See also: `TFPGList.First` (775), `TFPSList.Last` (805)

51.6.17 TFPGList.Items

Synopsis: Indexed access to items in the list.

Declaration: `Property Items[Index: Integer]: T; default`

Visibility: public

Access: Read,Write

Description: `Items` provides indexed access to the items in the list. The items can be get or set.

The index `Index` is zero based, and has a maximum value of `Count-1` (804).

Errors: If an invalid index is used, an `EListError` (767) exception is raised.

See also: `TFPSList.Count` (804)

51.6.18 TFPGList.List

Synopsis: Internal list object.

Declaration: `Property List : PTypeList`

Visibility: public

Access: Read

Description: `List` is the internal list of items. It should not be used directly.

See also: `TFPGList.Items` (775)

51.7 TFPGListEnumerator

51.7.1 Description

`TFPGListEnumerator` is a generic list enumerator. It is used in the `TFPGList` ([771](#)) class to implement the enumerator for the list.

Normally there should be no need to instantiate or use this class directly.

See also: `TFPGList` ([771](#))

51.7.2 Method overview

Page	Method	Description
776	<code>Create</code>	Create a new list enumerator.
776	<code>MoveNext</code>	Move to next element in the list.

51.7.3 Property overview

Page	Properties	Access	Description
776	<code>Current</code>	<code>r</code>	Current enumerated element.

51.7.4 TFPGListEnumerator.Create

Synopsis: Create a new list enumerator.

Declaration: `constructor Create (AList: TFPList)`

Visibility: `public`

Description: `Create` is called by the list `AList` to initialize a new enumerator. There should be no need to call this directly.

See also: `TFPGList` ([771](#))

51.7.5 TFPGListEnumerator.MoveNext

Synopsis: Move to next element in the list.

Declaration: `function MoveNext : Boolean`

Visibility: `public`

Description: `MoveNext` moves to the next element in the list.

See also: `TFPGListEnumerator.Current` ([776](#))

51.7.6 TFPGListEnumerator.Current

Synopsis: Current enumerated element.

Declaration: `Property Current : T`

Visibility: `public`

Access: `Read`

Description: `Current` returns the currently enumerated element. It is only valid after `TFPGListEnumerator.MoveNext` (776) was called and returned `True`.

See also: `TFPGListEnumerator.MoveNext` (776)

51.8 TFPGMap

51.8.1 Description

`TFPGMap` is a generic map class. It can be used to specialize a map for any key type and data type that do not require manual reference counting: For reference counted interface objects, `TFPGMap-InterfacedObjectData` (782) must be used.

See also: `TFPGMapInterfacedObjectData` (782)

51.8.2 Method overview

Page	Method	Description
778	<code>Add</code>	Add a key and value to the map.
778	<code>AddOrSetData</code>	Add data with given or set value if the key already exists.
777	<code>Create</code>	Create a new instance of the map.
778	<code>Find</code>	Find item based on key.
779	<code>IndexOf</code>	Find index of a key in the list.
779	<code>IndexOfData</code>	Find index of data value in the list.
779	<code>InsertKey</code>	Insert a new key in the list.
779	<code>InsertKeyData</code>	Insert a new key with associated data in the list.
780	<code>Remove</code>	Remove a key from the list.
778	<code>TryGetData</code>	Find data or return default.

51.8.3 Property overview

Page	Properties	Access	Description
780	<code>Data</code>	rw	Indexed access to the data in the list.
781	<code>KeyData</code>	rw	Access to data based on key.
780	<code>Keys</code>	rw	Indexed access to the keys in the list.
781	<code>OnCompare</code>	rw	Alias for <code>OnKeyCompare</code> .
782	<code>OnDataCompare</code>	rw	Compare function for data values.
781	<code>OnKeyCompare</code>	rw	Compare function for key values.

51.8.4 TFPGMap.Create

Synopsis: Create a new instance of the map.

Declaration: `constructor Create`

Visibility: `public`

Description: `Create` instantiates a new map. It mainly initializes the `TFPSMap` (806) parent with the sized of the key and data.

See also: `TFPSMap.Create` (806)

51.8.5 TFPGMap.Add

Synopsis: Add a key and value to the map.

Declaration: `function Add(const AKey: TKey; const AData: TData) : Integer`
`function Add(const AKey: TKey) : Integer`

Visibility: public

Description: Add adds a new key AKey of generic type TKey with data value AData to the list and returns the position at which the key was added.

Errors: If the item could not be added, an EListError (767) exception is raised. If Duplicates (809) is set to dupError and a duplicate key is added, an EListError (767) exception is raised.

See also: TFPGMap.Keys (780), TFPGMap.IndexOf (779), TFPGMap.KeyData (781), TFPGMap.Data (780), TFPSMap.Duplicates (809)

51.8.6 TFPGMap.Find

Synopsis: Find item based on key.

Declaration: `function Find(const AKey: TKey; out Index: Integer) : Boolean`

Visibility: public

Description: Find will search the key equal to AKey and return its index in AIndex. The return value of the function is True if an exact match for AKey is found, False otherwise.

The behaviour of Find is undefined if the map is not sorted. For unsorted maps, use IndexOf (779) instead.

See also: TFPGMap.IndexOf (779), TFPGMap.IndexOfData (779)

51.8.7 TFPGMap.TryGetData

Synopsis: Find data or return default.

Declaration: `function TryGetData(const AKey: TKey; out AData: TData) : Boolean`

Visibility: public

Description: TryGetData will search the map for AKey and return True or False depending on whether the value with the given key was found. If the key was found, the associated value is returned in AData, if it is not found a default value (using Default) is returned.

Errors: None.

See also: TFPGMap.Find (778)

51.8.8 TFPGMap.AddOrSetData

Synopsis: Add data with given or set value if the key already exists.

Declaration: `procedure AddOrSetData(const AKey: TKey; const AData: TData)`

Visibility: public

Description: AddOrSetData will check if key AKey already exists. if yes, the value associated with it will be replaced with AData. If the key does not yet exist, it will be added with value AData.

Errors: None.

See also: `TFPGMap.TryGetData` (778), `TFPGMap.Add` (778), `TFPGMap.Find` (778)

51.8.9 `TFPGMap.IndexOf`

Synopsis: Find index of a key in the list.

Declaration: `function IndexOf(const AKey: TKey) : Integer`

Visibility: public

Description: `IndexOf` returns the index of `AKey` in the list, or -1 if the key was not found in the list.

Errors: None.

See also: `TFPGMap.Find` (778), `TFPGMap.IndexOfData` (779)

51.8.10 `TFPGMap.IndexOfData`

Synopsis: Find index of data value in the list.

Declaration: `function IndexOfData(const AData: TData) : Integer`

Visibility: public

Description: `IndexOfData` returns the index of `AData` in the list, or -1 if the data was not found in the list.

Errors: None.

See also: `TFPGMap.Find` (778), `TFPGMap.IndexOf` (779)

51.8.11 `TFPGMap.InsertKey`

Synopsis: Insert a new key in the list.

Declaration: `procedure InsertKey(Index: Integer; const AKey: TKey)`

Visibility: public

Description: `InsertKey` inserts key `AKey` at position `Index` in the list. It is not allowed to insert a key in a sorted list.

Errors: If the index `AIndex` is out of range `[0..Count-1]`, or the list is sorted, an `EListError` (767) exception will be raised.

See also: `TFPGMap.InsertKeyData` (779), `TFPGMap.Add` (778), `TFPSMap.Delete` (806), `TFPSMap.Remove` (809)

51.8.12 `TFPGMap.InsertKeyData`

Synopsis: Insert a new key with associated data in the list.

Declaration: `procedure InsertKeyData(Index: Integer; const AKey: TKey;
const AData: TData)`

Visibility: public

Description: `InsertKey` inserts key `AKey` with associated data `AData` at position `Index` in the list. It is not allowed to insert a key in a sorted list.

Errors: If the index `AIndex` is out of range `[0..Count-1]`, or the list is sorted, an `EListError` (767) exception will be raised.

See also: `TFPGMap.InsertKey` (779), `TFPGMap.Add` (778), `TFPSMap.Delete` (806), `TFPGMap.Remove` (780)

51.8.13 `TFPGMap.Remove`

Synopsis: Remove a key from the list.

Declaration: `function Remove(const AKey: TKey) : Integer`

Visibility: public

Description: `Remove` removes the key `AKey` from the list, together with its associated data. The function returns the index of `AKey` prior to removal from the list, or -1 if `AKey` was not present in the list.

Errors: None.

See also: `TFPGMap.InsertKey` (779), `TFPGMap.InsertKeyData` (779), `TFPGMap.Add` (778), `TFPSMap.Delete` (806)

51.8.14 `TFPGMap.Keys`

Synopsis: Indexed access to the keys in the list.

Declaration: `Property Keys[Index: Integer]: TKey`

Visibility: public

Access: Read,Write

Description: `Keys` provides indexed access to the key values in the list. Valid values for `Index` are in the range `[0..Count-1]`. Key values can always be read, but can only be written if the list is unsorted.

Errors: If the index `AIndex` is out of range `[0..Count-1]`, an `EListError` (767) exception will be raised. The same exception is raised if a key is written and the list is sorted.

See also: `TFPSList.Count` (804), `TFPGMap.Data` (780), `TFPGMap.KeyData` (781)

51.8.15 `TFPGMap.Data`

Synopsis: Indexed access to the data in the list.

Declaration: `Property Data[Index: Integer]: TData`

Visibility: public

Access: Read,Write

Description: `Data` provides indexed access to the data values in the list. Valid values for `Index` are in the range `[0..Count-1]`. Data can always be read or written.

Errors: If the index `AIndex` is out of range `[0..Count-1]`, an `EListError` (767) exception will be raised.

See also: `TFPSList.Count` (804), `TFPGMap.Keys` (780), `TFPGMap.KeyData` (781)

51.8.16 TFPGMap.KeyData

Synopsis: Access to data based on key.

Declaration: `Property KeyData[AKey: TKey]: TData; default`

Visibility: public

Access: Read,Write

Description: `KeyData` allows access to the data based on the key value `AKey`. The data can be read and written. When writing, writing using an existing key will overwrite the current data. If it does not exist yet, it will be created. When reading, if the key is not present, an `EListError` (767) will be raised.

Errors: If the key does not exist, an `EListError` (767) exception will be raised.

See also: `TFPSList.Count` (804), `TFPGMap.Keys` (780), `TFPGMap.Data` (780)

51.8.17 TFPGMap.OnCompare

Synopsis: Alias for `OnKeyCompare`.

Declaration: `Property OnCompare : TKeyCompareFunc`

Visibility: public

Access: Read,Write

Description: `OnCompare` is a deprecated property, use `TFPGMap.OnKeyCompare` (781) instead.

See also: `TFPGMap.OnKeyCompare` (781)

51.8.18 TFPGMap.OnKeyCompare

Synopsis: Compare function for key values.

Declaration: `Property OnKeyCompare : TKeyCompareFunc`

Visibility: public

Access: Read,Write

Description: `OnKeyCompare` can be set to a function that compares key values. The default value for this event is a function that compares keys based on a byte-by-byte comparison of the memory block. The function must have the following semantics:

•Function signature:

```
function(const Key1, Key2: TKey): Integer;
```

- If the result of this function is negative, the first key (`Key1`) is assumed to be 'less' than the second key (`Key2`) and will be moved before the second in the list.
- If the function result is positive, the first key (`Key1`) pointer is assumed to be 'greater than' the second key (`Key2`) and will be moved after the second in the list.
- if the function result is zero, the keys are assumed to be 'equal' and no moving will take place.

See also: `TFPGMap.OnDataCompare` (782)

51.8.19 TFPGMap.OnDataCompare

Synopsis: Compare function for data values.

Declaration: `Property OnDataCompare : TDataCompareFunc`

Visibility: `public`

Access: `Read,Write`

Description: `OnDataCompare` can be set to a function that compares data values. The default value for this event is a function that compares data based on a byte-by-byte comparison of the memory block. The function must have the following semantics:

•Function signature:

```
function(const Data1, Data2: TData): Integer;
```

- If the result of this function is negative, the first data item (`Data1`) is assumed to be 'less' than the second data item (`Data2`) and will be moved before the second in the list.
- If the function result is positive, the first data item (`Data1`) pointer is assumed to be 'greater than' the second data item (`Data2`) and will be moved after the second in the list.
- if the function result is zero, the data items are assumed to be 'equal' and no moving will take place.

See also: `TFPGMap.OnKeyCompare` ([781](#))

51.9 TFPGMapInterfacedObjectData

51.9.1 Description

`TFPGInterfacedObjectMap` is a generic map class. It can be used to specialize a map for any key type, with associated data type that requires manual reference counting: any type which implements `IInterface`. For non-reference counted objects, `TFPGMap` ([777](#)) should be used.

This map class is entirely equivalent to `TFPGMap` ([777](#)), but operates on data items that require additional reference counting code on the data.

See also: `TFPGMap` ([777](#))

51.9.2 Method overview

Page	Method	Description
783	<code>Add</code>	Add a key and value to the map.
784	<code>AddOrSetData</code>	Add data with given or set value if the key already exists.
783	<code>Create</code>	Create a new instance of the map.
783	<code>Find</code>	Find item based on key.
784	<code>IndexOf</code>	Find index of a key in the list.
784	<code>IndexOfData</code>	Find index of data value in the list.
785	<code>InsertKey</code>	Insert a new key in the list.
785	<code>InsertKeyData</code>	Insert a new key with associated data in the list.
785	<code>Remove</code>	Remove a key from the list.
784	<code>TryGetData</code>	Find data or return default.

51.9.3 Property overview

Page	Properties	Access	Description
786	Data	rw	Indexed access to the data in the list.
786	KeyData	rw	Access to data based on key.
785	Keys	rw	Indexed access to the keys in the list.
786	OnCompare	rw	Alias for <code>OnKeyCompare</code> .
787	OnDataCompare	rw	Compare function for data values.
787	OnKeyCompare	rw	Compare function for key values.

51.9.4 TFPGMapInterfacedObjectData.Create

Synopsis: Create a new instance of the map.

Declaration: `constructor Create`

Visibility: `public`

Description: `Create` instantiates a new map. It mainly initializes the `TFPSMap` ([806](#)) parent with the sized of the key and data.

See also: `TFPSMap.Create` ([806](#))

51.9.5 TFPGMapInterfacedObjectData.Add

Synopsis: Add a key and value to the map.

Declaration: `function Add(const AKey: TKey; const AData: TData) : Integer`
`function Add(const AKey: TKey) : Integer`

Visibility: `public`

Description: `Add` adds a new key `AKey` of generic type `TKey` with data value `AData` to the list and returns the position at which the key was added.

Errors: If the item could not be added, an `EListError` ([767](#)) exception is raised. If `Duplicates` ([809](#)) is set to `dupError` and a duplicate key is added, an `EListError` ([767](#)) exception is raised.

See also: `TFPGMapInterfacedObjectData.Keys` ([785](#)), `TFPGMapInterfacedObjectData.IndexOf` ([784](#)), `TFPGMapInterfacedObjectData.KeyData` ([786](#)), `TFPGMapInterfacedObjectData.Data` ([786](#)), `TFPS.Duplicates` ([766](#))

51.9.6 TFPGMapInterfacedObjectData.Find

Synopsis: Find item based on key.

Declaration: `function Find(const AKey: TKey; out Index: Integer) : Boolean`

Visibility: `public`

Description: `Find` will search the first key smaller than or equal to `AKey` and return its index in `AIndex`. If the key was not found then -1 is returned. The return value of the function is `True` if an exact match for `AKey` is found, `False` otherwise.

See also: `TFPGMapInterfacedObjectData.IndexOf` ([784](#)), `TFPGMapInterfacedObjectData.IndexOfData` ([784](#))

51.9.7 TFPGMapInterfacedObjectData.TryGetData

Synopsis: Find data or return default.

Declaration: `function TryGetData(const AKey: TKey; out AData: TData) : Boolean`

Visibility: public

Description: `TryGetData` will search the map for `AKey` and return `True` or `False` depending on whether the value with the given key was found. If the key was found, the associated value is returned in `AData`, if it is not found a default value (using `Default`) is returned.

Errors: None.

See also: `TFPGMapInterfacedObjectData.Find` ([783](#))

51.9.8 TFPGMapInterfacedObjectData.AddOrSetData

Synopsis: Add data with given or set value if the key already exists.

Declaration: `procedure AddOrSetData(const AKey: TKey; const AData: TData)`

Visibility: public

Description: `AddOrSetData` will check if key `AKey` already exists. if yes, the value associated with it will be replaced with `AData`. If the key does not yet exist, it will be added with value `AData`.

Errors: None.

See also: `TFPGMapInterfacedObjectData.TryGetData` ([784](#)), `TFPGMapInterfacedObjectData.Add` ([783](#)), `TFPGMapInterfacedObjectData.Find` ([783](#))

51.9.9 TFPGMapInterfacedObjectData.IndexOf

Synopsis: Find index of a key in the list.

Declaration: `function IndexOf(const AKey: TKey) : Integer`

Visibility: public

Description: `IndexOf` returns the index of `AKey` in the list, or -1 if the key was not found in the list.

Errors: None.

See also: `TFPGMapInterfacedObjectData.Find` ([783](#)), `TFPGMapInterfacedObjectData.IndexOfData` ([784](#))

51.9.10 TFPGMapInterfacedObjectData.IndexOfData

Synopsis: Find index of data value in the list.

Declaration: `function IndexOfData(const AData: TData) : Integer`

Visibility: public

Description: `IndexOfData` returns the index of `AData` in the list, or -1 if the data was not found in the list.

Errors: None.

See also: `TFPGMapInterfacedObjectData.Find` ([783](#)), `TFPGMapInterfacedObjectData.IndexOf` ([784](#))

51.9.11 TFPGMapInterfacedObjectData.InsertKey

Synopsis: Insert a new key in the list.

Declaration: `procedure InsertKey(Index: Integer; const AKey: TKey)`

Visibility: public

Description: `InsertKey` inserts key `AKey` at position `Index` in the list. It is not allowed to insert a key in a sorted list.

Errors: If the index `AIndex` is out of range `[0..Count-1]`, or the list is sorted, an `EListError` (767) exception will be raised.

See also: `TFPGMapInterfacedObjectData.InsertKeyData` (785), `TFPGMapInterfacedObjectData.Add` (783), `TFPSMapInterfacedObjectData.Delete` (766), `TFPSMapInterfacedObjectData.Remove` (766)

51.9.12 TFPGMapInterfacedObjectData.InsertKeyData

Synopsis: Insert a new key with associated data in the list.

Declaration: `procedure InsertKeyData(Index: Integer; const AKey: TKey;
const AData: TData)`

Visibility: public

Description: `InsertKey` inserts key `AKey` with associated data `AData` at position `Index` in the list. It is not allowed to insert a key in a sorted list.

Errors: If the index `AIndex` is out of range `[0..Count-1]`, or the list is sorted, an `EListError` (767) exception will be raised.

See also: `TFPGMapInterfacedObjectData.InsertKey` (785), `TFPGMapInterfacedObjectData.Add` (783), `TFPSMapInterfacedObjectData.Delete` (766), `TFPGMapInterfacedObjectData.Remove` (785)

51.9.13 TFPGMapInterfacedObjectData.Remove

Synopsis: Remove a key from the list.

Declaration: `function Remove(const AKey: TKey) : Integer`

Visibility: public

Description: `Remove` removes the key `AKey` from the list, together with its associated data. The function returns the index of `AKey` prior to removal from the list, or -1 if `AKey` was not present in the list.

Errors: None.

See also: `TFPGMap.InsertKey` (779), `TFPGMap.InsertKeyData` (779), `TFPGMap.Add` (778), `TFPGMapInterfacedObjectData.Delete` (782)

51.9.14 TFPGMapInterfacedObjectData.Keys

Synopsis: Indexed access to the keys in the list.

Declaration: `Property Keys[Index: Integer]: TKey`

Visibility: public

Access: Read,Write

Description: `Keys` provides indexed access to the key values in the list. Valid values for `Index` are in the range `[0..Count-1]`. Key values can always be read, but can only be written if the list is unsorted.

Errors: If the index `AIndex` is out of range `[0..Count-1]`, an `EListError` (767) exception will be raised. The same exception is raised if a key is written and the list is sorted.

See also: `TFPSList.Count` (804), `TFPGMapInterfacedObjectData.Data` (786), `TFPGMapInterfacedObjectData.KeyData` (786)

51.9.15 TFPGMapInterfacedObjectData.Data

Synopsis: Indexed access to the data in the list.

Declaration: `Property Data[Index: Integer]: TData`

Visibility: public

Access: Read,Write

Description: `Data` provides indexed access to the data values in the list. Valid values for `Index` are in the range `[0..Count-1]`. Data can always be read or written.

Errors: If the index `AIndex` is out of range `[0..Count-1]`, an `EListError` (767) exception will be raised.

See also: `TFPSList.Count` (804), `TFPGMapInterfacedObjectData.Keys` (785), `TFPGMapInterfacedObjectData.KeyData` (786)

51.9.16 TFPGMapInterfacedObjectData.KeyData

Synopsis: Access to data based on key.

Declaration: `Property KeyData[AKey: TKey]: TData; default`

Visibility: public

Access: Read,Write

Description: `KeyData` allows access to the data based on the key value `AKey`. The data can be read and written. When writing, writing using an existing key will overwrite the current data. If it does not exist yet, it will be created. When reading, if the key is not present, an `EListError` (767) will be raised.

Errors: If the key does not exist, an `EListError` (767) exception will be raised.

See also: `TFPSList.Count` (804), `TFPGMapInterfacedObjectData.Keys` (785), `TFPGMapInterfacedObjectData.Data` (786)

51.9.17 TFPGMapInterfacedObjectData.OnCompare

Synopsis: Alias for `OnKeyCompare`.

Declaration: `Property OnCompare : TKeyCompareFunc`

Visibility: public

Access: Read,Write

Description: `OnCompare` is a deprecated property, use `TFPGMapInterfacedObjectData.OnKeyCompare` (787) instead.

See also: `TFPGMapInterfacedObjectData.OnKeyCompare` (787)

51.9.18 TFPGMapInterfacedObjectData.OnKeyCompare

Synopsis: Compare function for key values.

Declaration: `Property OnKeyCompare : TKeyCompareFunc`

Visibility: `public`

Access: `Read,Write`

Description: `OnKeyCompare` can be set to a function that compares key values. The default value for this event is a function that compares keys based on a byte-by-byte comparison of the memory block. The function must have the following semantics:

- Function signature:

```
function(const Key1, Key2: TKey): Integer;
```

- If the result of this function is negative, the first key (`Key1`) is assumed to be 'less' than the second key (`Key2`) and will be moved before the second in the list.
- If the function result is positive, the first key (`Key1`) pointer is assumed to be 'greater than' the second key (`Key2`) and will be moved after the second in the list.
- if the function result is zero, the keys are assumed to be 'equal' and no moving will take place.

See also: `TFPGMapInterfacedObjectData.OnDataCompare` ([787](#))

51.9.19 TFPGMapInterfacedObjectData.OnDataCompare

Synopsis: Compare function for data values.

Declaration: `Property OnDataCompare : TDataCompareFunc`

Visibility: `public`

Access: `Read,Write`

Description: `OnDataCompare` can be set to a function that compares data values. The default value for this event is a function that compares data based on a byte-by-byte comparison of the memory block. The function must have the following semantics:

- Function signature:

```
function(const Data1, Data2: TData): Integer;
```

- If the result of this function is negative, the first data item (`Data1`) is assumed to be 'less' than the second data item (`Data2`) and will be moved before the second in the list.
- If the function result is positive, the first data item (`Data1`) pointer is assumed to be 'greater than' the second data item (`Data2`) and will be moved after the second in the list.
- if the function result is zero, the data items are assumed to be 'equal' and no moving will take place.

See also: `TFPGMapInterfacedObjectData.OnKeyCompare` ([787](#))

51.10 TFPGMapObject

51.10.1 Description

`TFPGMapObject` is a generic map class. It can be used to specialize a map for object types, but not objects with automated reference counting: For reference counted interface objects, `TFPGMapInterfacedObjectData` (782) must be used.

See also: `TFPGMapInterfacedObjectData` (782)

51.10.2 Method overview

Page	Method	Description
788	<code>Add</code>	Add a key and value to the map.
789	<code>AddOrSetData</code>	Add data with given or set value if the key already exists.
788	<code>Create</code>	Create a new instance of the map.
789	<code>Find</code>	Find item based on key.
790	<code>IndexOf</code>	Find index of a key in the list.
790	<code>IndexOfData</code>	Find index of data value in the list.
790	<code>InsertKey</code>	Insert a new key in the list.
790	<code>InsertKeyData</code>	Insert a new key with associated data in the list.
791	<code>Remove</code>	Remove a key from the list.
789	<code>TryGetData</code>	Find data or return default.

51.10.3 Property overview

Page	Properties	Access	Description
791	<code>Data</code>	rw	Indexed access to the data in the list.
791	<code>KeyData</code>	rw	Access to data based on key.
791	<code>Keys</code>	rw	Indexed access to the keys in the list.
792	<code>OnCompare</code>	rw	Alias for <code>OnKeyCompare</code> .
792	<code>OnDataCompare</code>	rw	Compare function for data values.
792	<code>OnKeyCompare</code>	rw	Compare function for key values.

51.10.4 TFPGMapObject.Create

Synopsis: Create a new instance of the map.

Declaration: `constructor Create(AFreeObjects: Boolean)`
`constructor Create`

Visibility: `public`

Description: `Create` instantiates a new map. It mainly initializes the `TFPSMap` (806) parent with the sized of the key and data.

See also: `TFPSMap.Create` (806)

51.10.5 TFPGMapObject.Add

Synopsis: Add a key and value to the map.

Declaration: `function Add(const AKey: TKey; const AData: TData) : Integer`
`function Add(const AKey: TKey) : Integer`

Visibility: public

Description: Add adds a new key `AKey` of generic type `TKey` with data object `AData` to the list and returns the position at which the key was added.

Errors: If the item could not be added, an `EListError` (767) exception is raised. If `Duplicates` (809) is set to `dupError` and a duplicate key is added, an `EListError` (767) exception is raised.

See also: `TFPGMapObject.Keys` (791), `TFPGMapObject.IndexOf` (790), `TFPGMapObject.KeyData` (791), `TFPGMapObject.Data` (791), `TFPSMap.Duplicates` (809)

51.10.6 TFPGMapObject.Find

Synopsis: Find item based on key.

Declaration: `function Find(const AKey: TKey; out Index: Integer) : Boolean`

Visibility: public

Description: `Find` will search the first key smaller than or equal to `AKey` and return its index in `AIndex`. If the key was not found then -1 is returned. The return value of the function is `True` if an exact match for `AKey` is found, `False` otherwise.

See also: `TFPGMapObject.IndexOf` (790), `TFPGMapObject.IndexOfData` (790)

51.10.7 TFPGMapObject.TryGetData

Synopsis: Find data or return default.

Declaration: `function TryGetData(const AKey: TKey; out AData: TData) : Boolean`

Visibility: public

Description: `TryGetData` will search the map for `AKey` and return `True` or `False` depending on whether the value with the given key was found. If the key was found, the associated value is returned in `AData`, if it is not found a default value (using `Default`) is returned.

Errors: None.

See also: `TFPGMapObject.Find` (789)

51.10.8 TFPGMapObject.AddOrSetData

Synopsis: Add data with given or set value if the key already exists.

Declaration: `procedure AddOrSetData(const AKey: TKey; const AData: TData)`

Visibility: public

Description: `AddOrSetData` will check if key `AKey` already exists. if yes, the value associated with it will be replaced with `AData`. If the key does not yet exist, it will be added with value `AData`.

Errors: None.

See also: `TFPGMapObject.TryGetData` (789), `TFPGMapObject.Add` (788), `TFPGMapObject.Find` (789)

51.10.9 TFPGMapObject.IndexOf

Synopsis: Find index of a key in the list.

Declaration: `function IndexOf(const AKey: TKey) : Integer`

Visibility: public

Description: `IndexOf` returns the index of `AKey` in the list, or -1 if the key was not found in the list.

Errors: None.

See also: `TFPGMapObject.Find` (789), `TFPGMapObject.IndexOfData` (790)

51.10.10 TFPGMapObject.IndexOfData

Synopsis: Find index of data value in the list.

Declaration: `function IndexOfData(const AData: TData) : Integer`

Visibility: public

Description: `IndexOfData` returns the index of object `AData` in the list, or -1 if the data was not found in the list.

Errors: None.

See also: `TFPGMapObject.Find` (789), `TFPGMapObject.IndexOf` (790)

51.10.11 TFPGMapObject.InsertKey

Synopsis: Insert a new key in the list.

Declaration: `procedure InsertKey(Index: Integer; const AKey: TKey)`

Visibility: public

Description: `InsertKey` inserts key `AKey` at position `Index` in the list. It is not allowed to insert a key in a sorted list.

Errors: If the index `AIndex` is out of range `[0..Count-1]`, or the list is sorted, an `EListError` (767) exception will be raised.

See also: `TFPGMapObject.InsertKeyData` (790), `TFPGMapObject.Add` (788), `TFPSMap.Delete` (806), `TFPSMap.Remove` (809)

51.10.12 TFPGMapObject.InsertKeyData

Synopsis: Insert a new key with associated data in the list.

Declaration: `procedure InsertKeyData(Index: Integer; const AKey: TKey;
const AData: TData)`

Visibility: public

Description: `InsertKey` inserts key `AKey` with associated data `AData` at position `Index` in the list. It is not allowed to insert a key in a sorted list.

Errors: If the index `AIndex` is out of range `[0..Count-1]`, or the list is sorted, an `EListError` (767) exception will be raised.

See also: `TFPGMapObject.InsertKey` (790), `TFPGMapObject.Add` (788), `TFPSMap.Delete` (806), `TFPGMapObject.Remove` (791)

51.10.13 TFPGMapObject.Remove

Synopsis: Remove a key from the list.

Declaration: `function Remove(const AKey: TKey) : Integer`

Visibility: public

Description: `Remove` removes the key `AKey` from the list, together with its associated data. The function returns the index of `AKey` prior to removal from the list, or -1 if `AKey` was not present in the list.

Errors: None.

See also: `TFPGMapObject.InsertKey` (790), `TFPGMapObject.InsertKeyData` (790), `TFPGMapObject.Add` (788), `TFPSMap.Delete` (806)

51.10.14 TFPGMapObject.Keys

Synopsis: Indexed access to the keys in the list.

Declaration: `Property Keys[Index: Integer]: TKey`

Visibility: public

Access: Read,Write

Description: `Keys` provides indexed access to the key values in the list. Valid values for `Index` are in the range `[0..Count-1]`. Key values can always be read, but can only be written if the list is unsorted.

Errors: If the index `AIndex` is out of range `[0..Count-1]`, an `EListError` (767) exception will be raised. The same exception is raised if a key is written and the list is sorted.

See also: `TFPSList.Count` (804), `TFPGMapObject.Data` (791), `TFPGMapObject.KeyData` (791)

51.10.15 TFPGMapObject.Data

Synopsis: Indexed access to the data in the list.

Declaration: `Property Data[Index: Integer]: TData`

Visibility: public

Access: Read,Write

Description: `Data` provides indexed access to the data values in the list. Valid values for `Index` are in the range `[0..Count-1]`. Data can always be read or written.

Errors: If the index `AIndex` is out of range `[0..Count-1]`, an `EListError` (767) exception will be raised.

See also: `TFPSList.Count` (804), `TFPGMapObject.Keys` (791), `TFPGMapObject.KeyData` (791)

51.10.16 TFPGMapObject.KeyData

Synopsis: Access to data based on key.

Declaration: `Property KeyData[AKey: TKey]: TData; default`

Visibility: public

Access: Read,Write

Description: `KeyData` allows access to the data based on the key value `AKey`. The data can be read and written. When writing, writing using an existing key will overwrite the current data. If it does not exist yet, it will be created. When reading, if the key is not present, an `EListError` (767) will be raised.

Errors: If the key does not exist, an `EListError` (767) exception will be raised.

See also: `TFPSList.Count` (804), `TFPGMapObject.Keys` (791), `TFPGMapObject.Data` (791)

51.10.17 TFPGMapObject.OnCompare

Synopsis: Alias for `OnKeyCompare`.

Declaration: `Property OnCompare : TKeyCompareFunc`

Visibility: public

Access: Read,Write

Description: `OnCompare` is a deprecated property, use `TFPGMapObject.OnKeyCompare` (792) instead.

See also: `TFPGMapObject.OnKeyCompare` (792)

51.10.18 TFPGMapObject.OnKeyCompare

Synopsis: Compare function for key values.

Declaration: `Property OnKeyCompare : TKeyCompareFunc`

Visibility: public

Access: Read,Write

Description: `OnKeyCompare` can be set to a function that compares key values. The default value for this event is a function that compares keys based on a byte-by-byte comparison of the memory block. The function must have the following semantics:

- Function signature:

```
function(const Key1, Key2: TKey): Integer;
```

- If the result of this function is negative, the first key (`Key1`) is assumed to be 'less' than the second key (`Key2`) and will be moved before the second in the list.
- If the function result is positive, the first key (`Key1`) pointer is assumed to be 'greater than' the second key (`Key2`) and will be moved after the second in the list.
- if the function result is zero, the keys are assumed to be 'equal' and no moving will take place.

See also: `TFPGMapObject.OnDataCompare` (792)

51.10.19 TFPGMapObject.OnDataCompare

Synopsis: Compare function for data values.

Declaration: `Property OnDataCompare : TDataCompareFunc`

Visibility: public

Access: Read,Write

Description: `OnDataCompare` can be set to a function that compares data values. The default value for this event is a function that compares data based on a byte-by-byte comparison of the memory block. The function must have the following semantics:

- Function signature:

```
function(const Data1, Data2: TData): Integer;
```

- If the result of this function is negative, the first data item (`Data1`) is assumed to be 'less' than the second data item (`Data2`) and will be moved before the second in the list.
- If the function result is positive, the first data item (`Data1`) pointer is assumed to be 'greater than' the second data item (`Data2`) and will be moved after the second in the list.
- if the function result is zero, the data items are assumed to be 'equal' and no moving will take place.

See also: `TFPGMapObject.OnKeyCompare` ([792](#))

51.11 TFPGObjectList

51.11.1 Description

`TFPGList` can be used to specialize a list for any class type `T` that does not require reference counting (such as interfaced objects). It will specialize to a list with the same methods as `TFPSList` ([798](#)) or `classes.TFPList` ([766](#)) or `TFPObjectList`

See also: `TFPSList` ([798](#)), `classes.TFPList` ([766](#))

51.11.2 Method overview

Page	Method	Description
794	Add	Add new object of class <code>T</code> to the list.
795	AddList	Adds the elements from another list.
795	Assign	Copy objects from Source list.
794	Create	Instantiate a new object list.
794	Extract	Extract an item from the list.
794	GetEnumerator	Return a list enumerator for <code>T</code> .
795	IndexOf	Index of item.
795	Insert	Insert a new object in the list.
796	Remove	Remove an object from the list.
796	Sort	Sort the objects in the list.

51.11.3 Property overview

Page	Properties	Access	Description
796	First	rw	First non-nil item.
797	FreeObjects	rw	Does the list own the objects or not?
797	Items	rw	Indexed access to objects in the list.
797	Last	rw	Last non- <code>Nil</code> object.
797	List	r	Internal list pointer.

51.11.4 TFPGObjectList.Create

Synopsis: Instantiate a new object list.

Declaration: `constructor Create(FreeObjects: Boolean)`

Visibility: public

Description: `Create` instantiates a new object list. It will simply call the inherited constructor with the correct item size and will initialize `TFPGObjectList.FreeObjects` (797) with `FreeObjects`.

If `FreeObjects` is true, then the list owns the objects. Freeing or clearing the list will remove all objects from memory by calling the destructor. Deleting or removing an item from the list will also dispose of the element.

See also: `TFPGObjectList.FreeObjects` (797)

51.11.5 TFPGObjectList.Add

Synopsis: Add new object of class T to the list.

Declaration: `function Add(const Item: T) : Integer`

Visibility: public

Description: `Add` adds a new item `Item` of class type T to the list and returns the position at which the item was added.

Errors: If the item could not be added, an `EListError` (767) exception is raised.

See also: `TFPGObjectList.Extract` (794), `TFPGObjectList.Items` (797), `TFPGObjectList.IndexOf` (795)

51.11.6 TFPGObjectList.Extract

Synopsis: Extract an item from the list.

Declaration: `function Extract(const Item: T) : T`

Visibility: public

Description: `Extract` removes `Item` from the list and returns the removed item, or `Nil` if it was not found.

The extracted object will not be destroyed even if the list owns the objects.

Errors: None.

See also: `TFPSList.Delete` (800)

51.11.7 TFPGObjectList.GetEnumerator

Synopsis: Return a list enumerator for T.

Declaration: `function GetEnumerator : TFPGListEnumeratorSpec`

Visibility: public

Description: `GetEnumerator` returns an enumerator for the elements in the list. It is a specialized version of `TFPGListEnumerator` (776).

See also: `TFPGListEnumerator` (776)

51.11.8 TFPGObjectList.IndexOf

Synopsis: Index of item.

Declaration: `function IndexOf(const Item: T) : Integer`

Visibility: public

Description: `IndexOf` returns the index of `Item` in the list, or -1 if the item does not appear in the list.

Errors: None.

See also: `TFPGObjectList.Items` ([797](#)), `TFPGObjectList.Insert` ([795](#)), `TFPGObjectList.Add` ([794](#))

51.11.9 TFPGObjectList.Insert

Synopsis: Insert a new object in the list.

Declaration: `procedure Insert(Index: Integer; const Item: T)`

Visibility: public

Description: `Insert` inserts a new object (`Item`) in the list at position `Index`. The index is zero based and must be less than `Count` ([804](#)).

Errors: If an invalid index is specified, an `EListError` ([767](#)) exception is raised.

See also: `TFPGList.Items` ([775](#)), `TFPGList.Insert` ([773](#)), `TFPSList.Add` ([799](#))

51.11.10 TFPGObjectList.AddList

Synopsis: Adds the elements from another list.

Declaration: `procedure AddList(Source: TFPGObjectList)`

Visibility: public

Description: `AddList` adds all the elements from list `Source` to the current list.

See also: `TFPGList.AddList` ([774](#)), `TFPSList.AddList` ([803](#)), `TFPGObjectList.Add` ([794](#))

51.11.11 TFPGObjectList.Assign

Synopsis: Copy objects from Source list.

Declaration: `procedure Assign(Source: TFPGObjectList)`

Visibility: public

Description: `Assign` clears the list and copies all items in `Source` to the list. The source list must be of the same type as the destination list.

Take care if both the list owns the objects (i.e. have `TFPGObjectList.FreeObjects` ([797](#)) set to `True`), this may result to access violations.

See also: `TFPSList.Clear` ([800](#)), `TFPGObjectList.Add` ([794](#))

51.11.12 TFPGObjectList.Remove

Synopsis: Remove an object from the list.

Declaration: `function Remove(const Item: T) : Integer`

Visibility: public

Description: `Remove` removes the object `Item` from the list, and returns the index of the removed item. If no item was removed, `-1` is returned. Only the first object is removed.

If the list owns the objects, (`TFPGObjectList.FreeObjects` (797) is set to `True`) then the object is freed.

Errors: None.

See also: `TFPGObjectList.IndexOf` (795), `TFPSList.Delete` (800), `TFPGObjectList.FreeObjects` (797)

51.11.13 TFPGObjectList.Sort

Synopsis: Sort the objects in the list.

Declaration: `procedure Sort(Compare: TCompareFunc)`

Visibility: public

Description: `Sort` sorts the elements in the list using the provided `Compare` function. The list passes 2 items to the compare function. The result of this function determines how the items will be sorted:

- If the result of this function is negative, the first object (`Item1`) is assumed to be 'less' than the second object (`Item2`) and will be moved before the second in the list.
- If the function result is positive, the first object (`Item1`) is assumed to be 'greater than' the second object (`Item2`) and will be moved after the second in the list.
- if the function result is zero, the objects are assumed to be 'equal' and no moving will take place.

Errors: None.

51.11.14 TFPGObjectList.First

Synopsis: First non-nil item.

Declaration: `Property First : T`

Visibility: public

Access: Read, Write

Description: `First` returns the first non-nil item. If no such element is present, `Nil` is returned.

See also: `TFPSList.First` (805), `TFPGList.Last` (775), `TFPSList.Pack` (803)

51.11.15 TFPGObjectList.Last

Synopsis: Last non-`Nil` object.

Declaration: `Property Last : T`

Visibility: `public`

Access: `Read,Write`

Description: `Last` returns the last non-`Nil` object. If no such element is present, `Nil`. is returned.

See also: `TFPGObjectList.First` (796), `TFPSList.Last` (805)

51.11.16 TFPGObjectList.Items

Synopsis: Indexed access to objects in the list.

Declaration: `Property Items[Index: Integer]: T; default`

Visibility: `public`

Access: `Read,Write`

Description: `Items` provides indexed access to the objects in the list. The objects can be get or set.

The index `Index` is zero based, and has a maximum value of `Count-1` (804).

If the list owns the objects, (`TFPGObjectList.FreeObjects` (797) is set to `True`) then the previous object at position `Index` is freed when setting the property.

Errors: If an invalid index is used, an `EListError` (767) exception is raised.

See also: `TFPSList.Count` (804), `TFPGObjectList.FreeObjects` (797)

51.11.17 TFPGObjectList.List

Synopsis: Internal list pointer.

Declaration: `Property List : PTypeList`

Visibility: `public`

Access: `Read`

Description: `List` is the internal list of objects. It should not be used directly.

See also: `TFPGObjectList.Items` (797)

51.11.18 TFPGObjectList.FreeObjects

Synopsis: Does the list own the objects or not?

Declaration: `Property FreeObjects : Boolean`

Visibility: `public`

Access: `Read,Write`

Description: `FreeObjects` indicates whether the list owns the objects or not. If set to `True`, freeing or clearing the list will remove all objects from memory by calling the destructor. Deleting or removing an item from the list will also dispose of the element.

The initial value for this property is set in the constructor and is `True` by default.

See also: `TFPObjectList.FreeObjects` (797)

51.12 TFPSList

51.12.1 Description

`TFPSList` can be seen as the generalized equivalent of the classes unit `TFPList` (766) list. It is used as a base class for the `TFPGList` (771), `TFPGMap` (777), `TFPObjectList` (793), `TFPGInterfaceObjectList` (767) and `TFPGMapInterfacedObjectData` (782) generic classes.

This list is not meant to be used directly, it is an auxiliary class for the actual generic list classes.

See also: `classes.TFPList` (766), `TFPGMap` (777), `TFPObjectList` (793), `TFPGInterfaceObjectList` (767), `TFPGMapInterfacedObjectData` (782)

51.12.2 Method overview

Page	Method	Description
799	<code>Add</code>	Add a new item to the list.
803	<code>AddList</code>	Adds the elements from another list.
802	<code>Assign</code>	Copy one list to another.
800	<code>Clear</code>	Clear the list.
799	<code>Create</code>	Create a new instance of <code>TFPSList</code> .
800	<code>Delete</code>	Delete an item from the list.
800	<code>DeleteRange</code>	Delete a range of elements.
799	<code>Destroy</code>	Destroy the list instance.
800	<code>Error</code>	Raise an <code>EListError</code> exception.
801	<code>Exchange</code>	Exchange two items in the list.
801	<code>Expand</code>	Expand the capacity of the list.
801	<code>Extract</code>	delete an element from the list.
801	<code>IndexOf</code>	Search an item in the list.
802	<code>Insert</code>	Insert a new item in the list.
799	<code>ItemIsManaged</code>	Indicates whether the list item is a managed type or not.
802	<code>Move</code>	Moves an item from one position in the list to another.
803	<code>Pack</code>	Remove empty items from the list.
803	<code>Remove</code>	Remove the item from the list.
803	<code>Sort</code>	Sort the list.

51.12.3 Property overview

Page	Properties	Access	Description
804	<code>Capacity</code>	rw	Current capacity of the list.
804	<code>Count</code>	rw	Current element count.
805	<code>First</code>	rw	Pointer to first non-empty item in the list.
804	<code>Items</code>	rw	Items in the list.
805	<code>ItemSize</code>	r	Size of the items in the list.
805	<code>Last</code>	rw	Pointer to last non-empty item in the list.
805	<code>List</code>	r	Internal list pointer.

51.12.4 TFPSList.Create

Synopsis: Create a new instance of `TFPSList`.

Declaration: `constructor Create(AItemSize: Integer)`

Visibility: `public`

Description: `Create` creates a new instance of `TFPSList`, and initializes the item size to `ItemSize`, which defaults to the size of a pointer.

See also: `TFPSList.ItemSize` (805), `TFPSList.Destroy` (799)

51.12.5 TFPSList.Destroy

Synopsis: Destroy the list instance.

Declaration: `destructor Destroy; Override`

Visibility: `public`

Description: `Destroy` clears and cleans up the list instance. Depending on the descendant, this may also remove the items in the list from memory.

See also: `TFPSList.Clear` (800)

51.12.6 TFPSList.ItemsManaged

Synopsis: Indicates whether the list item is a managed type or not.

Declaration: `class function ItemIsManaged : Boolean; Virtual`

Visibility: `public`

Description: `ItemIsManaged` indicates whether the list item type is a managed type or not. If not, faster codepaths are sometimes taken in the code.

51.12.7 TFPSList.Add

Synopsis: Add a new item to the list.

Declaration: `function Add(Item: Pointer) : Integer`

Visibility: `public`

Description: `Add` adds the item pointed to by `Item` to the list. It is not the pointer `Item` itself which is added to the list, but rather the `TFPSList.ItemSize` (805) bytes of memory to which `Item` is pointing.

The function returns the index of the newly added item.

Errors: If the maximum list size is reached, an `EListError` (767) is raised.

See also: `TFPSList.Delete` (800), `TFPSList.Items` (804), `TFPSList.Clear` (800)

51.12.8 TFPSList.Clear

Synopsis: Clear the list.

Declaration: `procedure Clear`

Visibility: `public`

Description: `Clear` removes all the items in the list. Depending on the descendent, the list items themselves may be cleared as well.

See also: `TFPSList.Delete` (800), `TFPSList.Items` (804), `TFPSList.Add` (799)

51.12.9 TFPSList.Delete

Synopsis: Delete an item from the list.

Declaration: `procedure Delete(Index: Integer)`

Visibility: `public`

Description: `Delete` deletes the item at position `Index` from the list. Depending on the descendent, the list items itself may be cleared as well.

Errors: If `Index` is out of bounds, an `EListError` (767) exception is raised.

See also: `TFPSList.Clear` (800), `TFPSList.Items` (804), `TFPSList.Add` (799)

51.12.10 TFPSList.DeleteRange

Synopsis: Delete a range of elements.

Declaration: `procedure DeleteRange(IndexFrom: Integer; IndexTo: Integer)`

Visibility: `public`

Description: `DeleteRange` deletes elements from `IndexFrom` till `IndexTo`. Both indexes are zero based, and `IndexTo` must be bigger than `IndexFrom`. Using this method results in less moving of data in memory, and as such is more effective than deleting the elements one by one using `TFPSList.Delete` (800).

Errors: If invalid indexes are specified, an `EListError` (767) exception will be raised.

See also: `EListError` (767), `TFPSList.Delete` (800)

51.12.11 TFPSList.Error

Synopsis: Raise an `EListError` exception.

Declaration: `class procedure Error(const Msg: string; Data: PtrInt)`

Visibility: `public`

Description: `Error` is an auxiliary routine which raises an `EListError` (767) exception formatted from `Msg` and `Data`.

See also: `EListError` (767)

51.12.12 TFPSList.Exchange

Synopsis: Exchange two items in the list.

Declaration: `procedure Exchange (Index1: Integer; Index2: Integer)`

Visibility: public

Description: `Exchange` will exchange 2 items at positions `Index1` and `Index2` in the list.

Errors: If `Index1` or `Index2` are out of bounds, an `EListError` (767) exception is raised.

51.12.13 TFPSList.Expand

Synopsis: Expand the capacity of the list.

Declaration: `function Expand : TFPSList`

Visibility: public

Description: `Expand` will expand the capacity of the list, and returns itself.

Errors: If the size will become larger than `MaxListSize` (766) an `EListError` (767) exception will be raised.

See also: `MaxListSize` (766), `TFPSList.Capacity` (804)

51.12.14 TFPSList.Extract

Synopsis: delete an element from the list.

Declaration: `procedure Extract (Item: Pointer; ResultPtr: Pointer)`

Visibility: public

Description: `Extract` removes the item pointed to by `Item` from the list. It returns a pointer to the actually removed item from the list. The item is searched using `TFPSList.IndexOf` (801). If the item is not found, `nil` is returned.

Some descendants own the items in the list. `Extract` will not dispose of the item, as `TFPSList.Delete` (800) does.

Errors: None.

See also: `TFPSList.Delete` (800), `TFPSList.Add` (799), `TFPSList.IndexOf` (801)

51.12.15 TFPSList.IndexOf

Synopsis: Search an item in the list.

Declaration: `function IndexOf (Item: Pointer) : Integer`

Visibility: public

Description: `IndexOf` searches in the list for the item pointed to by `Item`, and returns the position (0-based index) of the item in the list, or -1 if it was not found. The items are compared using `sysutils.CompareMem` (766), so an exact memory layout match.

See also: `TFPSList.Extract` (801)

51.12.16 TFPSList.Insert

Synopsis: Insert a new item in the list.

Declaration: `procedure Insert (Index: Integer; Item: Pointer)`
`function Insert (Index: Integer) : Pointer`

Visibility: public

Description: `Insert` comes in 2 overloaded version. The version without `Item` creates a slot for a new item at position `Index` in the list, and returns a pointer to the new slot. The slot will be of size `TFPSList.ItemSize` (805).

The version with `Item` argument will allocate a slot in the list at position `Index` and will copy the item pointed to by `Item` to the slot in the list.

In both cases, `Index` must be 0-based.

Errors: If `Index` is invalid, an `EListError` (767) exception will be raised.

See also: `TFPSList.Add` (799), `TFPSList.Delete` (800), `TFPSList.Extract` (801)

51.12.17 TFPSList.Move

Synopsis: Moves an item from one position in the list to another.

Declaration: `procedure Move (CurIndex: Integer; NewIndex: Integer)`

Visibility: public

Description: `Move` moves the item at position `CurIndex` to position `NewIndex`. This is done by storing the value at position `CurIndex`, deleting the pointer at position `CurIndex`, and reinserting the value at position `NewIndex`

Errors: If `CurIndex` or `Newindex` are not inside the valid range of indices, an `EListError` (767) exception is raised.

See also: `Exchange` (801)

51.12.18 TFPSList.Assign

Synopsis: Copy one list to another.

Declaration: `procedure Assign (Obj: TFPSList)`

Visibility: public

Description: `Assign` clears the list and will add all items from `Obj` to the list. The items are copied one by one.

Errors: If the `TFPSList.ItemSize` (805) differs for the two lists, an `EListError` (767) exception is raised.

See also: `TFPSList.Add` (799)

51.12.19 TFPSList.AddList

Synopsis: Adds the elements from another list.

Declaration: `procedure AddList (Obj: TFPSList)`

Visibility: public

Description: `AddList` adds all the elements from list `Obj` to the current list. A check is done that the 2 lists have the same element size.

Errors: If the lists have different element size, an `EListError` (767) exception will be raised.

See also: `TFPSList.Add` (799), `EListError` (767)

51.12.20 TFPSList.Remove

Synopsis: Remove the item from the list.

Declaration: `function Remove (Item: Pointer) : Integer`

Visibility: public

Description: `Remove` searches `Item` in the list, and deletes it from the list. It returns the index of the item that was removed, or -1 if it was not found. Only the first match is removed.

If a descendent of `TFPSList` owns the object of the list, the item is removed from memory. If this is not desired, then `TFPSList.Extract` (801) must be used instead.

See also: `TFPSList.Extract` (801), `TFPSList.IndexOf` (801)

51.12.21 TFPSList.Pack

Synopsis: Remove empty items from the list.

Declaration: `procedure Pack`

Visibility: public

Description: `Pack` will remove all empty items from the list. An item is considered to be empty if the memory location where the item is stored contains only zero bytes.

See also: `TFPSList.Clear` (800), `TFPSList.Sort` (803)

51.12.22 TFPSList.Sort

Synopsis: Sort the list.

Declaration: `procedure Sort (Compare: TFPSListCompareFunc)`

Visibility: public

Description: `Sort` sorts the items in the list. Two pointers are compared by passing them to the `Compare` function. The result of this function determines how the pointers will be sorted:

- If the result of this function is negative, the first item is assumed to be 'less' than the second and will be moved before the second item in the list.
- If the function result is positive, the first item is assumed to be 'greater than' the second and will be moved after the second item in the list.

- if the function result is zero, the pointers are assumed to be 'equal' and no moving will take place.

The sort is done using a quicksort algorithm.

See also: [TFPSList.CompareFunc \(766\)](#)

51.12.23 TFPSList.Capacity

Synopsis: Current capacity of the list.

Declaration: `Property Capacity : Integer`

Visibility: public

Access: Read,Write

Description: `Capacity` is the current capacity (maximum amount of elements) of the list. The list capacity will expand automatically if an item is added and the capacity is reached (i.e. [TFPSList.Count \(804\)](#) equals `capacity`). Expanding the list is an expensive operation involving reallocation of memory and moving of list data in memory, so capacity can be set to a large amount to avoid frequent reallocations.

See also: [TFPSList.Count \(804\)](#), [TFPSList.Expand \(801\)](#), [TFPSList.Items \(804\)](#)

51.12.24 TFPSList.Count

Synopsis: Current element count.

Declaration: `Property Count : Integer`

Visibility: public

Access: Read,Write

Description: `Count` is the current amount of elements in the list. It is initially zero. A valid item index is always a value between zero and `Count-1`.

See also: [TFPSList.Items \(804\)](#), [TFPSList.Capacity \(804\)](#)

51.12.25 TFPSList.Items

Synopsis: Items in the list.

Declaration: `Property Items[Index: Integer]: Pointer; default`

Visibility: public

Access: Read,Write

Description: `Items` provides indexed access to the items in the list, the returned pointers are not the actual list items, but pointers to the elements in the list. The items can be get or set.

When assigning to the `Items` property, the memory area pointed to by the assigned pointer is copied to the list. Exactly [TFPSList.ItemSize \(805\)](#) bytes are copied.

The index `Index` is zero based, and has a maximum value of `Count-1` ([804](#)).

Errors: If an invalid index is used, an [EListError \(767\)](#) exception is raised.

See also: [TFPSList.ItemSize \(805\)](#), [TFPSList.Count \(804\)](#)

51.12.26 TFPSList.ItemSize

Synopsis: Size of the items in the list.

Declaration: `Property ItemSize : Integer`

Visibility: `public`

Access: `Read`

Description: `ItemSize` is the memory size of the items in the list. It is specified in the constructor and cannot be changed during the lifetime of the list.

See also: `TFPSList.Create` ([799](#))

51.12.27 TFPSList.List

Synopsis: Internal list pointer.

Declaration: `Property List : PByte`

Visibility: `public`

Access: `Read`

Description: `List` is a pointer to the memory area used for the items in the list. It should not be manipulated.

51.12.28 TFPSList.First

Synopsis: Pointer to first non-empty item in the list.

Declaration: `Property First : Pointer`

Visibility: `public`

Access: `Read,Write`

Description: `First` returns the value of the first non-empty item in the list. An item is considered empty if consists of `TFPSList.ItemSize` ([805](#)) zero bytes.

If there are no non-empty items in the list, then `Nil` is returned.

See also: `TFPSList.Last` ([805](#)), `TFPSList.Pack` ([803](#))

51.12.29 TFPSList.Last

Synopsis: Pointer to last non-empty item in the list.

Declaration: `Property Last : Pointer`

Visibility: `public`

Access: `Read,Write`

Description: `Last` returns the value of the last non-empty item in the list. An item is considered empty if consists of `TFPSList.ItemSize` ([805](#)) zero bytes.

If there are no non-empty items in the list, then `Nil` is returned.

See also: `TFPSList.Last` ([805](#)), `TFPSList.Pack` ([803](#))

51.13 TFPSMap

51.13.1 Description

TFPSMap can be used to create a map for any type `T` that does not require reference counting (such as interfaced objects). It will specialize to a map which is a generalized list with an arbitrary type as the list index (called the key).

This class should normally not be used directly, instead use one of the generic map objects such as TFPGMap (777).

See also: TFPGMap (777)

51.13.2 Method overview

Page	Method	Description
807	Add	Add a key, value pair to the map.
806	Create	Create a new map with given key and data size.
807	Find	Find data using the associated key.
807	IndexOf	Index of key pointed to by AKey.
808	IndexOfData	Index of data item AData.
808	Insert	Insert a new slot for key and associated data item in the list.
808	InsertKey	Insert a key in the list.
808	InsertKeyData	Insert a key and associated in the list.
809	Remove	Remove a key/value pair from the map.
809	Sort	Sort the list according to key.

51.13.3 Property overview

Page	Properties	Access	Description
810	Data	rw	Indexed access to the locations of all data items.
810	DataSize	r	Size (in bytes) for the data associated with keys.
809	Duplicates	rw	What to do with duplicate key values.
811	KeyData	rw	Access to data locations using key.
810	Keys	rw	Indexed access to the locations of all keys.
810	KeySize	r	Size (in bytes) for the key.
812	OnDataPtrCompare	rw	Callback to compare 2 data items.
811	OnKeyPtrCompare	rw	Callback to compare 2 keys.
811	OnPtrCompare	rw	Alias for OnKeyPtrCompare.
811	Sorted	rw	Is the map permanently sorted on key ?

51.13.4 TFPSMap.Create

Synopsis: Create a new map with given key and data size.

Declaration: `constructor Create(AKeySize: Integer; ADataSize: Integer)`

Visibility: `public`

Description: `Create` instantiates a new TFPSMap instance and initializes TFPSMap.KeySize (810) and TFPSMap.DataSize (810) with `AKeySize` and `ADataSize`, respectively. It also initializes the TFPSMap.OnDataPtrCompare (812) and TFPSMap.OnKeyPtrCompare (811) properties to functions that compare memory blocks.

Errors: None.

See also: [TFPSMap.Destroy \(806\)](#), [TFPSMap.KeySize \(810\)](#), [TFPSMap.DataSize \(810\)](#), [TFPSMap.OnDataPtrCompare \(812\)](#), [TFPSMap.OnKeyPtrCompare \(811\)](#)

51.13.5 TFPSMap.Add

Synopsis: Add a key, value pair to the map.

Declaration: `function Add(AKey: Pointer; AData: Pointer) : Integer`
`function Add(AKey: Pointer) : Integer`

Visibility: public

Description: `Add` adds the memory pointed to by `AData` to the map using the memory pointed to by `AKey` as the key. If no data is specified, it allocates a slot for `AKey` and returns a pointer to that slot.

Errors: If the maximum amount of values is reached, `Add` will raise an [EListError \(767\)](#) exception.

See also: [TFPSMap.Insert \(808\)](#), [TFPSMap.IndexOf \(807\)](#)

51.13.6 TFPSMap.Find

Synopsis: Find data using the associated key.

Declaration: `function Find(AKey: Pointer; out Index: Integer) : Boolean`

Visibility: public

Description: `Find` searches for the first key less than or equal to `AKey` and returns its index in the list in `Index`. It returns `True` if an exact match for the key was found. It returns `False` if the key was not found, and `Index` is then set to `-1`.

This function performs a binary search using the key comparing function specified in [OnKeyPtrCompare \(811\)](#).

Errors: if [OnKeyPtrCompare](#) is not set, an access violation will occur.

See also: [TFPSMap.OnKeyPtrCompare \(811\)](#)

51.13.7 TFPSMap.IndexOf

Synopsis: Index of key pointed to by `AKey`.

Declaration: `function IndexOf(AKey: Pointer) : Integer`

Visibility: public

Description: `IndexOf` returns the index of the element with a key pointed to by `AKey`. It returns `-1` if the key was not found.

If the list is sorted, then a binary search is performed, otherwise a linear search is used to find the key.

Errors: None.

See also: [TFPSMap.Find \(807\)](#), [TFPSMap.IndexOfData \(808\)](#), [TFPSMap.Keys \(810\)](#), [TFPSMap.Data \(810\)](#)

51.13.8 TFPSMap.IndexOfData

Synopsis: Index of data item AData.

Declaration: `function IndexOfData (AData: Pointer) : Integer`

Visibility: public

Description: `IndexOfData` returns the index of the element with data pointed to by AData. It returns -1 if the data item was not found. The search is always performed using a linear search.

See also: `TFPSMap.Find` (807), `TFPSMap.IndexOf` (807), `TFPSMap.Keys` (810), `TFPSMap.Data` (810)

51.13.9 TFPSMap.Insert

Synopsis: Insert a new slot for key and associated data item in the list.

Declaration: `function Insert (Index: Integer) : Pointer`
`procedure Insert (Index: Integer; out AKey: Pointer; out AData: Pointer)`

Visibility: public

Description: `Insert` will allocate a new slot in the list. It returns a pointer to the new slot. If AKey and AData are given, then they will point to the positions in the slot for the key and data items.

Errors: If the maximum amount of values is reached or an invalid index is specified, `Insert` will raise an `EListError` (767) exception.

See also: `TFPSMap.Add` (807), `TFPSMap.InsertKey` (808), `TFPSMap.InsertKeyData` (808)

51.13.10 TFPSMap.InsertKey

Synopsis: Insert a key in the list.

Declaration: `procedure InsertKey (Index: Integer; AKey: Pointer)`

Visibility: public

Description: `InsertKey` will allocate a new slot in the list for a key value as pointed to by AKey, and copy the key pointed to by AKey to the slot.

Errors: If the maximum amount of values is reached or an invalid index is specified, `InsertKey` will raise an `EListError` (767) exception.

See also: `TFPSMap.Add` (807), `TFPSMap.Insert` (808), `TFPSMap.InsertKeyData` (808)

51.13.11 TFPSMap.InsertKeyData

Synopsis: Insert a key and associated in the list.

Declaration: `procedure InsertKeyData (Index: Integer; AKey: Pointer; AData: Pointer)`

Visibility: public

Description: `InsertKeyData` will allocate a new slot in the list for a key value as pointed to by AKey, and copy the key pointed to by AKey as well as the data pointed to by AData to the newly allocated slot.

Errors: If the maximum amount of values is reached or an invalid index is specified, `InsertKeyData` will raise an `EListError` (767) exception.

See also: `TFPSMap.Add` (807), `TFPSMap.Insert` (808), `TFPSMap.InsertKey` (808)

51.13.12 TFPSMap.Remove

Synopsis: Remove a key/value pair from the map.

Declaration: `function Remove(AKey: Pointer) : Integer`

Visibility: `public`

Description: `Remove` removes the key/value pair pointing to by `AKey` from the map. It returns the index of `Akey` prior to removal from the list. If `AKey` was not found, -1 is returned.

Errors: None.

See also: `TFPSList.Delete` (800), `TFPSMap.IndexOf` (807)

51.13.13 TFPSMap.Sort

Synopsis: Sort the list according to key.

Declaration: `procedure Sort`

Visibility: `public`

Description: `Sort` sorts the list according to the key value, using the compare function provided in `TFPSMap.OnKeyPtrCompare` (811)

Errors: None.

See also: `TFPSMap.OnKeyPtrCompare` (811), `TFPSMap.OnDataPtrCompare` (812)

51.13.14 TFPSMap.Duplicates

Synopsis: What to do with duplicate key values.

Declaration: `Property Duplicates : TDuplicates`

Visibility: `public`

Access: Read,Write

Description: `Duplicates` can be set to determine what to do with duplicate key values in the map:

dupIgnoreIgnore the new item, do not add it to the list.

dupAcceptAccept duplicates, adding them to the list.

dupErrorRaise an error when an attempt is made to add a duplicate.

The value is ignored if `Sorted` (811) is `False`.

See also: `TFPSMap.Sorted` (811), `types.TDuplicates` (766)

51.13.15 TFPSMap.KeySize

Synopsis: Size (in bytes) for the key.

Declaration: `Property KeySize : Integer`

Visibility: `public`

Access: `Read`

Description: `KeySize` is the size (in bytes) for the keys in the map. This size is initialized during list construction and defaults to the size of a pointer.

See also: `TFPSMap.Create` (806), `TFPSMap.DataSize` (810), `TFPSMap.Keys` (810), `TFPSMap.KeyData` (811)

51.13.16 TFPSMap.DataSize

Synopsis: Size (in bytes) for the data associated with keys.

Declaration: `Property DataSize : Integer`

Visibility: `public`

Access: `Read`

Description: `DataSize` is the size (in bytes) for the data in the map. This size is initialized during list construction and defaults to the size of a pointer.

See also: `TFPSMap.Create` (806), `TFPSMap.KeySize` (810), `TFPSMap.Keys` (810), `TFPSMap.KeyData` (811), `TFPSMap.Data` (810)

51.13.17 TFPSMap.Keys

Synopsis: Indexed access to the locations of all keys.

Declaration: `Property Keys[Index: Integer]: Pointer`

Visibility: `public`

Access: `Read,Write`

Description: `Keys` provides indexed access to all keys. It does not return the key, but returns a pointer to the key value. When setting the `Keys` property, a pointer to the key value must be set, and the key value is copied from the pointer.

See also: `TFPSMap.KeySize` (810), `TFPSMap.KeyData` (811), `TFPSMap.Data` (810)

51.13.18 TFPSMap.Data

Synopsis: Indexed access to the locations of all data items.

Declaration: `Property Data[Index: Integer]: Pointer`

Visibility: `public`

Access: `Read,Write`

Description: `Data` provides indexed access to all data. It does not return the actual data, but returns a pointer to the data. When setting the `Data` property, a pointer to the data value must be set, and the data value is copied from the pointer.

See also: `TFPSMap.DataSize` (810), `TFPSMap.KeyData` (811), `TFPSMap.Keys` (810)

51.13.19 TFPSMap.KeyData

Synopsis: Access to data locations using key.

Declaration: `Property KeyData[Key: Pointer]: Pointer; default`

Visibility: public

Access: Read,Write

Description: `KeyData` provides access to the data items, using their key value (as pointed to by `AKey`) as an index. When reading a non-existent key value, `Nil` is returned. If the key is found, a pointer to the associated data's location is returned. When writing, the key pointed to by `Key` is added if it was not present, and the data data is copied from the written pointer.

See also: `TFPSMap.DataSize` (810), `TFPSMap.KeyData` (811), `TFPSMap.Data` (810), `TFPSMap.Keys` (810), `TFPSMap.KeySize` (810)

51.13.20 TFPSMap.Sorted

Synopsis: Is the map permanently sorted on key ?

Declaration: `Property Sorted : Boolean`

Visibility: public

Access: Read,Write

Description: `Sorted` can be set to true to keep the map permanently sorted on key value. The sorting happens using `TFPSMap.OnKeyPtrCompare` (811).

See also: `TFPSMap.OnKeyPtrCompare` (811), `TFPSMap.Sort` (809), `TFPSMap.Duplicates` (809)

51.13.21 TFPSMap.OnPtrCompare

Synopsis: Alias for `OnKeyPtrCompare`.

Declaration: `Property OnPtrCompare : TFPSListCompareFunc`

Visibility: public

Access: Read,Write

Description: `OnPtrCompare` is a deprecated alias for `OnKeyPtrCompare` (811).

See also: `TFPSMap.OnKeyPtrCompare` (811), `TFPSMap.OnDataPtrCompare` (812)

51.13.22 TFPSMap.OnKeyPtrCompare

Synopsis: Callback to compare 2 keys.

Declaration: `Property OnKeyPtrCompare : TFPSListCompareFunc`

Visibility: public

Access: Read,Write

Description: `OnKeyPtrCompare` is used to compare the values of 2 keys. By default it simply compares the byte values of the key memory block. It can be set to any function that performs another comparison. (e.g. a function that treats the memory blocks as a string pointer and compare the actual strings).

This function is used to sort the list or find a key.

See also: `TFPSListCompareFunc` (766), `TFPSMap.OnDataPtrCompare` (812), `TFPSMap.Sort` (809), `TFPSMap.Find` (807), `TFPSMap.IndexOf` (807)

51.13.23 TFPSMap.OnDataPtrCompare

Synopsis: Callback to compare 2 data items.

Declaration: `Property OnDataPtrCompare : TFPSListCompareFunc`

Visibility: public

Access: Read,Write

Description: `OnKeyPtrCompare` is used to compare the values of 2 keys. By default it simply compares the byte values of the key memory block. It can be set to any function that performs another comparison. (e.g. a function that treats the memory blocks as a string pointer and compare the actual strings).

This function is used to find a data item (`IndexOf` (808)).

See also: `TFPSListCompareFunc` (766), `TFPSMap.OnKeyPtrCompare` (811), `TFPSMap.IndexOfData` (808)

Chapter 52

Reference for unit 'fpwiderstring'

52.1 Used units

Table 52.1: Used units by unit 'fpwiderstring'

Name	Page
System	1335
unicodedata	2150

52.2 Overview

`fpwiderstring` implements Unicode string support for the Free Pascal RTL using native Object Pascal routines. It is meant to be used on operating systems where the operating system does not natively support Unicode transformations and operations.

In general, it is sufficient to include the unit in the uses clause of a program. The initialization code of the unit will set the Unicode string manager of the system unit to the object pascal implementation contained in this unit.

This unit needs Unicode collation and character set tables in order to be able to do its work correctly. These must be registered using the routines of the `unicodedata` ([813](#)) unit: the FPC project distributes some Unicode collation data in `.bco` files which can be loaded using the `LoadCollation` ([2161](#)) routine.

In order for sorting and comparing of strings to work, a collation must be used. The collation in general depends on the internationalization of the application. Since the system unit does not know about collations, the collation must be set in the `fpWideString` unit using the `SetActiveCollation` ([814](#)) function. The collation can be set on a per-thread basis.

New threads get `DefaultCollationName` ([814](#)) as the active collation name.

The `fpwiderstring` unit performs conversions between Unicode and single-byte ansistring conversions (excluding UTF8). Support for various single-byte encodings are based on the `charset` ([813](#)) unit. This unit can be used to load single byte code pages. Various code page units such as `cp895`, `cp932`, `cp950` are provided by the "rtl-unicode" package.

The `fpwiderstring` requires at least the Default Unicode Collation Element Table to be registered (called `DUCET`). The `DUCET` encoding is provided by the `unicodeducet` unit. More information can be found in the `unicodedata` ([813](#)) unit.

52.3 Constants, types and variables

52.3.1 Variables

`DefaultCollationName` : `UnicodeString` = ''

`DefaultCollationName` is the collation name for new threads. It is empty by default.

52.4 Procedures and functions

52.4.1 GetActiveCollation

Synopsis: Return the currently active collation for the current thread.

Declaration: `function GetActiveCollation : PUCA_DataBook`

Visibility: default

Description: `GetActiveCollation` returns the currently active collation, for the current thread, as set using the `SetActiveCollation` (814) function.

New threads get `DefaultCollationName` (814) as the active collation name. The collation can be changed per thread using the `SetActiveCollation` (814) function.

Errors: None.

See also: `SetActiveCollation` (814), `DefaultCollationName` (814)

52.4.2 SetActiveCollation

Synopsis: Set the active collation for the current thread.

Declaration: `function SetActiveCollation(const AName: UnicodeString) : Boolean`
`function SetActiveCollation(const ACollation: PUCA_DataBook) : Boolean`

Visibility: default

Description: `SetActiveCollation` sets the collation used in the current thread of the application. This can be done using the name of the collation (`AName`) as registered or in the `unicodedata` (813) unit or using the actual collation data (`ACollation`).

Errors: If the collation `AName` was not found, or `ACollation` is nil, then `False` is returned.

See also: `GetActiveCollation` (814), `DefaultCollationName` (814)

Chapter 53

Reference for unit 'getopts'

53.1 Used units

Table 53.1: Used units by unit 'getopts'

Name	Page
System	1335

53.2 Overview

This document describes the GETOPTS unit for Free Pascal. It was written for Linux by Michael Van Canneyt. It now also works for all supported platforms.

The getopts unit provides a mechanism to handle command-line options in a structured way, much like the GNU getopts mechanism. It allows you to define the valid options for your program, and the unit will then parse the command-line options for you, and inform you of any errors.

53.3 Constants, types and variables

53.3.1 Constants

`EndOfOptions = #255`

Returned by `getopt` ([817](#)), `getlongopts` ([816](#)) to indicate that there are no more options.

`No_Argument = 0`

Specifies that a long option does not take an argument.

`Optional_Argument = 2`

Specifies that a long option optionally takes an argument.

`OptSpecifier : Set of Char = ['-']`

Character indicating an option on the command-line.

`Required_Argument = 1`

Specifies that a long option needs an argument.

53.3.2 Types

`Orderings = (require_order, permute, return_in_order)`

Table 53.2: Enumeration values for type `Orderings`

Value	Explanation
<code>permute</code>	Change command-line options.
<code>require_order</code>	Don't touch the ordering of the command-line options.
<code>return_in_order</code>	Return options in the correct order.

Command-line ordering options.

`POption = ^TOption`

Pointer to `TOption` (820) record.

53.3.3 Variables

`OptArg : string`

Set to the argument of an option, if the option needs one.

`OptErr : Boolean`

Indicates whether `getopt()` prints error messages.

`OptInd : LongInt`

when all options have been processed, `optind` is the index of the first non-option parameter. This is a read-only variable. Note that it can become equal to `paramcount+1`.

`OptOpt : Char`

In case of an error, contains the character causing the error.

53.4 Procedures and functions

53.4.1 GetLongOpts

Synopsis: Return next long option.

Declaration: `function GetLongOpts (ShortOpts: string; LongOpts: POption;
var Longind: LongInt) : Char`

Visibility: default

Description: Returns the next option found on the command-line, taking into account long options as well. If no more options are found, returns `EndOfOptions`. If the option requires an argument, it is returned in the `OptArg` variable.

if any of the option definitions in `LongOpts` array has the `Flag` pointer set, then the return value is the null character (`#0` or `char(0)`) and the actual option letter is written in the location pointed to by `Flag`.

`ShortOptions` is a string containing all possible one-letter options. (see [Getopt \(817\)](#) for its description and use) `LongOpts` is a pointer to the first element of an array of `Option` records, the last of which needs a name of zero length.

The function tries to match the names even partially (i.e. `-app` will match e.g. the append option), but will report an error in case of ambiguity.

If the option needs an argument, set `Has_arg` to `Required_argument` (1), if the option optionally has an argument, set `Has_arg` to `Optional_argument` (2). If the option needs no argument, set `Has_arg` to zero.

Required arguments can be specified in two ways :

1. Pasted to the option : `-option=value`
2. As a separate argument : `-option value`

Optional arguments can only be specified through the first method.

Errors: see [Getopt \(817\)](#).

See also: [Getopt \(817\)](#)

53.4.2 GetOpt

Synopsis: Get next short option.

Declaration: `function GetOpt (ShortOpts: string) : Char`

Visibility: default

Description: Returns the next option found on the command-line. If no more options are found, returns `EndOfOptions`. If the option requires an argument, it is returned in the `OptArg` variable.

`ShortOptions` is a string containing all possible one-letter options. If a letter is followed by a colon (:), then that option needs an argument. If a letter is followed by 2 colons, the option has an optional argument. If the first character of `shortoptions` is a '+' then options following a non-option are regarded as non-options (standard Unix behavior). If it is a '-', then all non-options are treated as arguments of a option with character #0. This is useful for applications that require their options in the exact order as they appear on the command-line. If the first character of `shortoptions` is none of the above, options and non-options are permuted, so all non-options are behind all options. This allows options and non-options to be in random order on the command line.

Errors: Errors are reported through giving back a '?' character. `OptOpt` then gives the character which caused the error. If `OptErr` is `True` then `getopt` prints an error-message to `stdout`.

See also: [GetLongOpts \(816\)](#)

Listing: `./examples/optex/optex.pp`

```

program testopt;

{ Program to depmonstrate the getopt function. }

{
  Valid calls to this program are
  optex --verbose --add me --delete you
  optex --append --create child
  optex -ab -c me -d you
  and so on
}
uses getopt;

var c : char;
    optionindex : Longint;
    theopts : array[1..7] of TOption;

begin
  with theopts[1] do
    begin
      name:='add';
      has_arg:=1;
      flag:=nil;
      value:=#0;
    end;
  with theopts[2] do
    begin
      name:='append';
      has_arg:=0;
      flag:=nil;
      value:=#0;
    end;
  with theopts[3] do
    begin
      name:='delete';
      has_arg:=1;
      flag:=nil;
      value:=#0;
    end;
  with theopts[4] do
    begin
      name:='verbose';
      has_arg:=0;
      flag:=nil;
      value:=#0;
    end;
  with theopts[5] do
    begin
      name:='create';
      has_arg:=1;
      flag:=nil;
      value:='c'
    end;
  with theopts[6] do
    begin
      name:='file';
      has_arg:=1;
      flag:=nil;

```

```

    value:=#0;
end;
with theopts[7] do
begin
    name:="";
    has_arg:=0;
    flag:=nil;
end;
c:=#0;
repeat
    c:=getlongopts('abc:d:012',@theargs[1],optionindex);
    case c of
        '1','2','3','4','5','6','7','8','9' :
            begin
                writeln ('Got optind : ',c)
            end;
        #0 : begin
                write ('Long option : ',theargs[optionindex].name);
                if theargs[optionindex].has_arg>0 then
                    writeln (' With value : ',optarg)
                else
                    writeln
                end;
            'a' : writeln ('Option a. ');
            'b' : writeln ('Option b. ');
            'c' : writeln ('Option c : ', optarg);
            'd' : writeln ('Option d : ', optarg);
            '?' : writeln ('Error with opt : ',optopt);
        end; { case }
    until c=endofoptions;
    if optind<=paramcount then
        begin
            write ('Non options : ');
            while optind<=paramcount do
                begin
                    write (paramstr(optind),' ');
                    inc(optind)
                end;
            writeln
        end
    end.

```

53.5 TOption

```

TOption = record
public
    Name : string;
    Has_arg : Integer;
    Flag
    : PChar;
    Value : Char;
    procedure SetOption(const aName: string
    ; AHas_Arg: Integer;
                        AFlag: PChar; AValue: Char
    );

```

end

The `TOption` type is used to communicate the long options to `GetLongOpts` (816). The `Name` field is the name of the option. `Has_arg` specifies if the option wants an argument, `Flag` is a pointer to a char, which is set to `Value`, if it is non-nil.

53.5.1 Method overview

Page	Method	Description
820	<code>SetOption</code>	Set all option fields in 1 call.

53.5.2 `TOption.SetOption`

Synopsis: Set all option fields in 1 call.

Declaration: `procedure SetOption(const aName: string; AHas_Arg: Integer;
AFlag: PChar; AValue: Char)`

Visibility: public

Description: `SetOption` can be used to set all fields in 1 single call:

- Name is set to `aName`
- Has_arg is set to `aHas_arg`
- Flag is set to `aFlag`
- Value is set to `aValue`

Chapter 54

Reference for unit 'go32'

54.1 Used units

Table 54.1: Used units by unit 'go32'

Name	Page
System	1335

54.2 Overview

This document describes the GO32 unit for the Free Pascal compiler under dos. It was donated by Thomas Schatzl (tom_at_work@geocities.com), for which my thanks. This unit was first written for dos by Florian Klaempfl.

Only the GO32V2 DPMI mode is discussed by me here due to the fact that new applications shouldn't be created with the older GO32V1 model. The go32v2 version is much more advanced and better. Additionally a lot of functions only work in DPMI mode anyway. I hope the following explanations and introductions aren't too confusing at all. If you notice an error or bug send it to the FPC mailing list or directly to me. So let's get started and happy and error free coding I wish you.... Thomas Schatzl, 25. August 1998

54.3 Real mode callbacks.

The callback mechanism can be thought of as the converse of calling a real mode procedure (i.e. interrupt), which allows your program to pass information to a real mode program, or obtain services from it in a manner that's transparent to the real mode program. In order to make a real mode callback available, you must first get the real mode callback address of your procedure and the selector and offset of a register data structure. This real mode callback address (this is a segment:offset address) can be passed to a real mode program via a software interrupt, a dos memory block or any other convenient mechanism. When the real mode program calls the callback (via a far call), the DPMI host saves the registers contents in the supplied register data structure, switches into protected mode, and enters the callback routine with the following settings:

- interrupts disabled

- `%CS : %EIP` = 48 bit pointer specified in the original call to `get_rm_callback` (847)
- `%DS : %ESI` = 48 bit pointer to real mode `SS : SP`
- `%ES : %EDI` = 48 bit pointer of real mode register data structure.
- `%SS : %ESP` = locked protected mode stack
- All other registers undefined

The callback procedure can then extract its parameters from the real mode register data structure and/or copy parameters from the real mode stack to the protected mode stack. Recall that the segment register fields of the real mode register data structure contain segment or paragraph addresses that are not valid in protected mode. Far pointers passed in the real mode register data structure must be translated to virtual addresses before they can be used with a protected mode program. The callback procedure exits by executing an `IRET` with the address of the real mode register data structure in `%ES : %EDI`, passing information back to the real mode caller by modifying the contents of the real mode register data structure and/or manipulating the contents of the real mode stack. The callback procedure is responsible for setting the proper address for resumption of real mode execution into the real mode register data structure; typically, this is accomplished by extracting the return address from the real mode stack and placing it into the `%CS : %EIP` fields of the real mode register data structure. After the `IRET`, the DPMI host switches the CPU back into real mode, loads ALL registers with the contents of the real mode register data structure, and finally returns control to the real mode program. All variables and code touched by the callback procedure **MUST** be locked to prevent page faults.

See also: `get_rm_callback` (847), `free_rm_callback` (842), `lock_code` (858), `lock_data` (858)

54.4 Executing software interrupts.

Simply execute a `realintr()` call with the desired interrupt number and the supplied register data structure. But some of these interrupts require you to supply them a pointer to a buffer where they can store data to or obtain data from in memory. These interrupts are real mode functions and so they only can access the first Mb of linear address space, not FPC's data segment. For this reason FPC supplies a pre-initialized dos memory location within the GO32 unit. This buffer is internally used for dos functions too and so it's contents may change when calling other procedures. It's size can be obtained with `tb_size` (868) and it's linear address via `transfer_buffer` (869). Another way is to allocate a completely new dos memory area via the `global_dos_alloc` (854) function for your use and supply its real mode address.

See also: `tb_size` (868), `transfer_buffer` (869), `global_dos_alloc` (854), `global_dos_free` (856), `realintr` (860)

Listing: `./examples/go32ex/softint.pp`

{ Executes a real mode software interrupt

Exactly the interrupt call to get the DOS version.

get DOS version Int 21h / function 30h

Input:

AH = \$30

AL = \$1

Return:

AL = major version number

AH = minor version number

}

```

uses
    go32;

var
    r : treatregs;

begin
    r.ah := $30;
    r.al := $01;
    realintr($21, r);
    Writeln('DOS v', r.al, '.', r.ah, ' detected');
end.

```

Listing: ./examples/go32ex/rmpmint.pp

{ This example shows the difference between protected and real mode interrupts; it redirects the protected mode handler to an own handler which returns an impossible function result and calls it afterwards. Then the real mode handler is called directly, to show the difference between the two.

Used Interrupt:
get DOS version Int 21h / function 30h
Input: AH = \$30
AL = \$1
Return: AL = major version number
AH = minor version number
}

```

uses
    crt,
    go32;

var
    r : treatregs;
    { temporary variable used for the protected mode int call }
    axreg : Word;

    oldint21h : tseginfo;
    newint21h : tseginfo;

```

{ this is our int 21h protected mode interrupt handler. It catches the function call to get the DOS version, all other int 21h calls are redirected to the old handler; it is written in assembly because the old handler can't be called with pascal }

```

procedure int21h_handler; assembler;
asm

```

```

    cmpw $0x3001, %ax
    jne .LCallOld
    movw $0x3112, %ax
    iret

```

```

.LCallOld:
    ljmp %cs:oldint21h

```

```

end;

```

{ a small helper procedure, which waits for a keypress }

```

procedure resume;

```



```

begin
    Writeln;
    Write('— press any key to resume —'); readkey;
    gotoxy(1, wherey); clrcol;
end;

begin
    { see the text messages for further detail }
    clrscr;
    Writeln('Executing real mode interrupt');
    resume;
    r.ah := $30; r.al := $01; realintr($21, r);
    Writeln('DOS v', r.al, '.', r.ah, ' detected');
    resume;
    Writeln('Executing protected mode interrupt without our own',
        ' handler');
    Writeln;
    asm
        movb $0x30, %ah
        movb $0x01, %al
        int $0x21
        movw %ax, axreg
    end;
    Writeln('DOS v', r.al, '.', r.ah, ' detected');
    resume;
    Writeln('As you can see the DPML hosts default protected mode',
        ' handler');
    Writeln('simply redirects it to the real mode handler');
    resume;
    Writeln('Now exchanging the protected mode interrupt with our ',
        ' own handler');
    resume;

    newint21h.offset := @int21h_handler;
    newint21h.segment := get_cs;
    get_pm_interrupt($21, oldint21h);
    set_pm_interrupt($21, newint21h);

    Writeln('Executing real mode interrupt again');
    resume;
    r.ah := $30; r.al := $01; realintr($21, r);
    Writeln('DOS v', r.al, '.', r.ah, ' detected');
    Writeln;
    Writeln('See, it didn"t change in any way.");
    resume;
    Writeln('Now calling protected mode interrupt');
    resume;
    asm
        movb $0x30, %ah
        movb $0x01, %al
        int $0x21
        movw %ax, axreg
    end;
    Writeln('DOS v', lo(axreg), '.', hi(axreg), ' detected');
    Writeln;
    Writeln('Now you can see that there"s a distinction between ',
        ' the two ways of calling interrupts...');
    set_pm_interrupt($21, oldint21h);

```

end.

54.5 Software interrupts.

Ordinarily, a handler installed with `set_pm_interrupt` (865) only services software interrupts that are executed in protected mode; real mode software interrupts can be redirected by `set_rm_interrupt` (867).

See also: `set_rm_interrupt` (867), `get_rm_interrupt` (851), `set_pm_interrupt` (865), `get_pm_interrupt` (846), `lock_data` (858), `lock_code` (858), `enable` (841), `disable` (839), `outportb` (859)

54.6 Hardware interrupts.

Hardware interrupts are generated by hardware devices when something unusual happens; this could be a keypress or a mouse move or any other action. This is done to minimize CPU time, else the CPU would have to check all installed hardware for data in a big loop (this method is called 'polling') and this would take much time. A standard IBM-PC has two interrupt controllers, that are responsible for these hardware interrupts: both allow up to 8 different interrupt sources (IRQs, interrupt requests). The second controller is connected to the first through IRQ 2 for compatibility reasons, e.g. if controller 1 gets an IRQ 2, he hands the IRQ over to controller 2. Because of this up to 15 different hardware interrupt sources can be handled. IRQ 0 through IRQ 7 are mapped to interrupts 8h to Fh and the second controller (IRQ 8 to 15) is mapped to interrupt 70h to 77h. All of the code and data touched by these handlers MUST be locked (via the various locking functions) to avoid page faults at interrupt time. Because hardware interrupts are called (as in real mode) with interrupts disabled, the handler has to enable them before it returns to normal program execution. Additionally a hardware interrupt must send an EOI (end of interrupt) command to the responsible controller; this is accomplished by sending the value 20h to port 20h (for the first controller) or A0h (for the second controller). The following example shows how to redirect the keyboard interrupt.

Listing: `./examples/go32ex/keyclick.pp`

{ This example demonstrates how to chain to a hardware interrupt.

In more detail, it hooks the keyboard interrupt, calls a user procedure which in this case simply turns the PC speaker on and off. Then the old interrupt is called.

```
}

```

```
{ $ASMMODE ATT }
{ $MODE FPC }
```

uses

```
crt,
go32;
```

const

```
{ keyboard is IRQ 1 -> interrupt 9 }
kbdint = $9;
```

var

```
{ holds old PM interrupt handler address }
oldint9_handler : tseginfo;
{ new PM interrupt handler }
```

```

newint9_handler : tseinfo;

{ pointer to interrupt handler }
clickproc : pointer;
{ the data segment selector }
backupDS : Word; external name '___v2prt0_ds_alias';

{ interrupt handler }
procedure int9_handler; assembler;
asm
    cli
    { save all registers, because we don't know which the compiler
    uses for the called procedure }
    pushl %ds
    pushl %es
    pushl %fs
    pushl %gs
    pushal
    { set up to call a FPC procedure }
    movw %cs:backupDS, %ax
    movw %ax, %ds
    movw %ax, %es
    movw dosmemselector, %ax
    movw %ax, %fs
    { call user procedure }
    call *clickproc
    { restore all registers }
    popal
    popl %gs
    popl %fs
    popl %es
    popl %ds
    { note: in go32v2 mode %cs=%ds=%es !!! }
    jmp %cs:oldint9_handler { call old handler }
    { we don't need to do anything more, because the old interrupt
    handler does this for us (send EOI command, iret, sti...) }
end;
{ dummy procedure to retrieve exact length of handler, for locking
and unlocking functions }
procedure int9_dummy; begin end;

{ demo user procedure, simply clicks on every keypress }
procedure clicker;
begin
    sound(500); delay(10); nosound;
end;
{ dummy procedure to retrieve exact length of user procedure for
locking and unlocking functions }
procedure clicker_dummy; begin end;

{ installs our new handler }
procedure install_click;
begin
    clickproc := @clicker;
    { lock used code and data }
    lock_data(clickproc, sizeof(clickproc));
    lock_data(dosmemselector, sizeof(dosmemselector));

```

```

lock_code(@clicker,
    longint(@clicker_dummy) - longint(@clicker));
lock_code(@int9_handler,
    longint(@int9_dummy)-longint(@int9_handler));
{ fill in new handler's 48 bit pointer }
newint9_handler.offset := @int9_handler;
newint9_handler.segment := get_cs;
{ get old PM interrupt handler }
get_pm_interrupt(kbdint, oldint9_handler);
{ set the new interrupt handler }
set_pm_interrupt(kbdint, newint9_handler);
end;

{ deinstalls our interrupt handler }
procedure remove_click;
begin
    { set old handler }
    set_pm_interrupt(kbdint, oldint9_handler);
    { unlock used code & data }
    unlock_data(dosmemselector, sizeof(dosmemselector));
    unlock_data(clickproc, sizeof(clickproc));

    unlock_code(@clicker,
        longint(@clicker_dummy)-longint(@clicker));
    unlock_code(@int9_handler,
        longint(@int9_dummy)-longint(@int9_handler));
end;

var
    ch : char;

begin
    install_click;
    Writeln('Enter any message. Press return when finished');
    while (ch <> #13) do begin
        ch := readkey; write(ch);
    end;
    remove_click;
end.

```

54.7 Disabling interrupts.

The GO32 unit provides the two procedures `disable()` and `enable()` to disable and enable all interrupts.

54.8 Creating your own interrupt handlers.

Interrupt redirection with FPC pascal is done via the `set_pm_interrupt()` for protected mode interrupts or via the `set_rm_interrupt()` for real mode interrupts.

54.9 Protected mode interrupts vs. Real mode interrupts.

As mentioned before, there's a distinction between real mode interrupts and protected mode interrupts; the latter are protected mode programs, while the former must be real mode programs. To call a protected mode interrupt handler, an assembly 'int' call must be issued, while the other is called via the `realintr()` or `intr()` function. Consequently, a real mode interrupt then must either reside in dos memory (<1MB) or the application must allocate a real mode callback address via the `get_rm_callback()` function.

54.10 Handling interrupts with DPMI.

The interrupt functions are real-mode procedures; they normally can't be called in protected mode without the risk of an protection fault. So the DPMI host creates an interrupt descriptor table for the application. Initially all software interrupts (except for int 31h, 2Fh and 21h function 4Ch) or external hardware interrupts are simply directed to a handler that reflects the interrupt in real-mode, i.e. the DPMI host's default handlers switch the CPU to real-mode, issue the interrupt and switch back to protected mode. The contents of general registers and flags are passed to the real mode handler and the modified registers and flags are returned to the protected mode handler. Segment registers and stack pointer are not passed between modes.

54.11 Interrupt redirection.

Interrupts are program interruption requests, which in one or another way get to the processor; there's a distinction between software and hardware interrupts. The former are explicitly called by an 'int' instruction and are a bit comparable to normal functions. Hardware interrupts come from external devices like the keyboard or mouse. Functions that handle hardware interrupts are called handlers.

54.12 Processor access.

These are some functions to access various segment registers (`%cs`, `%ds`, `%ss`) which makes your work a bit easier.

See also: `get_cs` (842), `get_ds` (843), `get_ss` (853)

54.13 I/O port access.

The I/O port access is done via the various `inportb` (857), `outportb` (859) functions which are available. Additionally Free Pascal supports the Turbo Pascal `PORT[]`-arrays but it is by no means recommended to use them, because they're only for compatibility purposes.

See also: `outportb` (859), `inportb` (857)

54.14 dos memory access.

Dos memory is accessed by the predefined `dosmemselector` selector; the GO32 unit additionally provides some functions to help you with standard tasks, like copying memory from heap to dos memory and the likes. Because of this it is strongly recommended to use them, but you are still free

to use the provided standard memory accessing functions which use 48 bit pointers. The third, but only thought for compatibility purposes, is using the `mem[]`-arrays. These arrays map the whole 1 Mb dos space. They shouldn't be used within new programs. To convert a segment:offset real mode address to a protected mode linear address you have to multiply the segment by 16 and add its offset. This linear address can be used in combination with the `DOSMEMSELECTOR` variable.

See also: `dosmemget` (830), `dosmempnt` (831), `dosmemmove` (831), `dosmemfillchar` (830), `dosmemfillword` (830), `seg_move` (864), `seg_fillchar` (862), `seg_fillword` (863)

54.15 FPC specialities.

The `%ds` and `%es` selector MUST always contain the same value or some system routines may crash when called. The `%fs` selector is preloaded with the `DOSMEMSELECTOR` variable at startup, and it MUST be restored after use, because again FPC relies on this for some functions. Luckily we asm programmers can still use the `%gs` selector for our own purposes, but for how long ?

See also: `get_cs` (842), `get_ds` (843), `get_ss` (853), `allocate_ldt_descriptors` (834), `free_ldt_descriptor` (841), `segment_to_descriptor` (862), `get_next_selector_increment_value` (845), `get_segment_base_address` (853), `set_segment_base_address` (867), `set_segment_limit` (867), `create_code_segment_alias_descriptor` (839)

54.16 Selectors and descriptors.

Descriptors are a bit like real mode segments; they describe (as the name implies) a memory area in protected mode. A descriptor contains information about segment length, its base address and the attributes of it (i.e. type, access rights, ...). These descriptors are stored internally in a so-called descriptor table, which is basically an array of such descriptors. Selectors are roughly an index into this table. Because these 'segments' can be up to 4 GB in size, 32 bits aren't sufficient anymore to describe a single memory location like in real mode. 48 bits are now needed to do this, a 32 bit address and a 16 bit sized selector. The GO32 unit provides the `tseginfo` record to store such a pointer. But due to the fact that most of the time data is stored and accessed in the `%ds` selector, FPC assumes that all pointers point to a memory location of this selector. So a single pointer is still only 32 bits in size. This value represents the offset from the data segment base address to this memory location.

54.17 What is DPMI.

The dos Protected Mode Interface helps you with various aspects of protected mode programming. These are roughly divided into descriptor handling, access to dos memory, management of interrupts and exceptions, calls to real mode functions and other stuff. Additionally it automatically provides swapping to disk for memory intensive applications. A DPMI host (either a Windows dos box or `CWSDPMI.EXE`) provides these functions for your programs.

54.18 Constants, types and variables

54.18.1 Constants

```
auxcarryflag = $010
```

Check for auxiliary carry flag in `trealregs` (834).

```
carryflag = $001
```

Check for carry flag in `trealregs` (834).

```
directionflag = $400
```

Check for direction flag in `trealregs` (834).

```
dosmemfillchar : procedure(seg: Word; ofs: Word; count: LongInt; c
    : Char) = @ dpmi_dosmemfillchar
```

Sets a region of dos memory to a specific byte value.

Parameters:

seg real mode segment.

ofs real mode offset.

count number of bytes to set.

c value to set memory to.

Notes: No range check is performed.

```
dosmemfillword : procedure(seg: Word; ofs: Word; count: LongInt; w
    : Word) = @ dpmi_dosmemfillword
```

Sets a region of dos memory to a specific word value.

Parameters:

seg real mode segment.

ofs real mode offset.

count number of words to set.

w value to set memory to.

Notes: No range check is performed.

```
dosmemget : procedure(seg: Word; ofs: Word; var data; count: LongInt
    ) = @ dpmi_dosmemget
```

Copies data from the dos memory onto the heap.

Parameters:

seg source real mode segment.

ofs source real mode offset.

data destination.

count number of bytes to copy.

Notes: No range checking is performed.

For an example, see `global_dos_alloc` (854).

```
dosmemmove : procedure(sseg: Word; sofs: Word; dseg: Word; dofs: Word
;
count: LongInt) = @ dpmi_dosmemmove
```

Copies count bytes of data between two dos real mode memory locations.

Parameters:

sseg source real mode segment.

sofs source real mode offset.

dseg destination real mode segment.

dofs destination real mode offset.

count number of bytes to copy.

Notes: No range check is performed in any way.

```
dosmemput : procedure(seg: Word; ofs: Word; var data; count: LongInt
) = @ dpmi_dosmemput
```

Copies heap data to dos real mode memory.

Parameters:

seg destination real mode segment.

ofs destination real mode offset.

data source.

count number of bytes to copy.

Notes: No range checking is performed.

For an example, see `global_dos_alloc` (854).

```
interruptflag = $200
```

Check for interrupt flag in `trealregs` (834).

```
overflowflag = $800
```

Check for overflow flag in `trealregs` (834).

```
parityflag = $004
```

Check for parity flag in `trealregs` (834).


```
rm_dpml = 4
```

get_run_mode (852) return value: DPML (e.g. dos box or 386Max).

```
rm_raw = 1
```

get_run_mode (852) return value: raw (without HIMEM).

```
rm_unknown = 0
```

get_run_mode (852) return value: Unknown runmode.

```
rm_vcpi = 3
```

get_run_mode (852) return value: VCPI (with HIMEM and EMM386).

```
rm_xms = 2
```

get_run_mode (852) return value: XMS (with HIMEM, without EMM386).

```
signflag = $080
```

Check for sign flag in trealregs (834).

```
trapflag = $100
```

Check for trap flag in trealregs (834).

```
zeroflag = $040
```

Check for zero flag in trealregs (834).

54.18.2 Types

```
registers = trealregs
```

Alias for trealregs (834).

```
trealregs = record
case Integer of
1: (
    EDI : LongInt;
    ESI : LongInt
    ;
    EBP : LongInt;
    Res : LongInt;
    EBX : LongInt;
    EDX : LongInt
    ;
    ECX : LongInt;
    EAX : LongInt;
```

```
Flags : Word;
ES : Word;
DS : Word;
FS : Word;
GS : Word;
IP : Word;
CS : Word;
SP : Word;
SS : Word;
);
2: (
  DI : Word;
  DI2 : Word;
  SI
  : Word;
  SI2 : Word;
  BP : Word;
  BP2 : Word;
  R1 : Word;
  R2
  : Word;
  BX : Word;
  BX2 : Word;
  DX : Word;
  DX2 : Word;
  CX
  : Word;
  CX2 : Word;
  AX : Word;
  AX2 : Word;
);
3: (
  stuff
  : Array[1..4] of LongInt;
  BL : Byte;
  BH : Byte;
  BL2 : Byte
  ;
  BH2 : Byte;
  DL : Byte;
  DH : Byte;
  DL2 : Byte;
  DH2 : Byte
  ;
  CL : Byte;
  CH : Byte;
  CL2 : Byte;
  CH2 : Byte;
  AL : Byte
  ;
  AH : Byte;
  AL2 : Byte;
  AH2 : Byte;
);
```

```

4: (
    RealEDI : LongInt
    ;
    RealESI : LongInt;
    RealEBP : LongInt;
    RealRES : LongInt;
    RealEBX : LongInt;
    RealEDX : LongInt;
    RealECX : LongInt;
    RealEAX
    : LongInt;
    RealFlags : Word;
    RealES : Word;
    RealDS : Word;
    RealFS : Word;
    RealGS : Word;
    RealIP : Word;
    RealCS : Word
    ;
    RealSP : Word;
    RealSS : Word;
);
end

```

The `trealregs` type contains the data structure to pass register values to a interrupt handler or real mode callback.

54.18.3 Variables

```
dosmemselector : Word
```

Selector to the dos memory. The whole dos memory is automatically mapped to this single descriptor at startup. This selector is the recommended way to access dos memory.

```
int31error : Word
```

This variable holds the result of a DPMI interrupt call. Any nonzero value must be treated as a critical failure.

54.19 Procedures and functions

54.19.1 `allocate_ldt_descriptors`

Synopsis: Allocate a number of descriptors.

Declaration: `function allocate_ldt_descriptors(count: Word) : Word`

Visibility: default

Description: Allocates a number of new descriptors.

Parameters:

count specifies the number of requested unique descriptors.

Return value: The base selector.

Remark Notes: The descriptors allocated must be initialized by the application with other function calls. This function returns descriptors with a limit and size value set to zero. If more than one descriptor was requested, the function returns a base selector referencing the first of a contiguous array of descriptors. The selector values for subsequent descriptors in the array can be calculated by adding the value returned by the `get_next_selector_increment_value` (845) function.

Errors: Check the `int31error` (834) variable.

See also: `free_ldt_descriptor` (841), `get_next_selector_increment_value` (845), `segment_to_descriptor` (862), `create_code_segment_alias_descriptor` (839), `set_segment_limit` (867), `set_segment_base_address` (867)

Listing: `./examples/go32ex/seldes.pp`

{
This example demonstrates the usage of descriptors and the effects of changing its limit and base address.

In more detail, the program fills the region described by an allocated descriptor in text screen memory with various characters. Before doing this it saves the entire screen contents to the heap and restores it afterwards.

Some additional background:

The text screen of a VGA card has it's address space at \$B800:0; screen memory is organized in a linear fashion, e.g. the second line comes directly after the first, where each cell occupies 2 bytes of memory (1 byte character data, 1 byte attributes). It is 32 kb in size.

Hence the offset of a single memory cell from its origin is:

$$Y * columns * 2 + X * 2$$

where X and Y mark the point and columns is the number of character cells per line

}
 {\$mode delphi}

uses

crt,
 go32;

const

{ screen x and y dimensions }
 maxx = 80;
 maxy = 25;
 { bytes used for every character cell }
 bytespercell = 2;
 { screen size in bytes }
 screensize = maxx * maxy * bytespercell;

{ the linear address of \$B800:0 }
 linB8000 = \$B800 * 16;

type

string80 = **string**[80];

```

var
    { holds the old screen contents }
    text_save : array[0..screenSize-1] of byte;
    { old cursor x and y coordinates }
    text_oldx, text_oldy : Word;

    { selector to the text mode screen }
    text_sel : Word;

{ prints a status message on the first line of the screen and then
waits for a keypress }
procedure status(s : string80);
begin
    gotoxy(1, 1); clreol; write(s); readkey;
end;

{ writes some descriptor info on the last 2 lines }
procedure selinfo(sel : Word);
begin
    gotoxy(1, 24);
    clreol; writeln('Descriptor base address : $',
        hexstr(get_segment_base_address(sel), 8));
    clreol; write('Descriptor limit : ', get_segment_limit(sel));
end;

{ returns a 2 byte character cell, which includes character data
and its color attributes }
function makechar(ch : char; color : byte) : Word;
begin
    result := byte(ch) or (color shl 8);
end;

begin
    { save original screen contents to variable, this time by using
    seg_move() and the dosmemselector variable }
    seg_move(dosmemselector, linB8000, get_ds, longint(@text_save),
        screenSize);
    { additionally we have to save the old screen cursor
    coordinates }
    text_oldx := wherex; text_oldy := wherey;
    { clear the whole screen }
    seg_fillword(dosmemselector, linB8000, screenSize div 2,
        makechar(' ', Black or (Black shl 4)));
    { output message }
    status('Creating selector "text_sel" to a part of ' +
        'text screen memory');
    { allocate descriptor }
    text_sel := allocate_ldt_descriptors(1);
    { set its base address to the linear address of the text screen
    + the byte size of one line (=maxx * bytespercell * 1) }
    set_segment_base_address(text_sel,
        linB8000 + bytespercell * maxx * 1);
    { the limit is set to the screenSize reduced by one (a must be)
    and the number of lines we don't want to have touched (first
    line + lower 2 lines) }
    set_segment_limit(text_sel, screenSize - 1 - bytespercell *
        maxx * 3);

```

```

{ write descriptor info }
selinfo(text_sel);

status('and clearing entire memory selected by "text_sel" +
      ' descriptor');
{ fill the entire selected memory with single characters }
seg_fillword(text_sel, 0, (get_segment_limit(text_sel)+1) div 2,
             makechar(' ', LightBlue shl 4));

status('Notice that only the memory described by the ' +
      'descriptor changed, nothing else');

status('Now reducing it"s limit and base and setting it"s ' +
      'described memory');
{ set the base address of the descriptor (increase it by the
  byte size of one line) }
set_segment_base_address(text_sel,
    get_segment_base_address(text_sel) + bytespercell * maxx);
{ decrease the limit by byte size of 2 lines (1 line because
  base address changed, one line on the lower end) }
set_segment_limit(text_sel,
    get_segment_limit(text_sel) - bytespercell * maxx * 2);
{ write descriptor info }
selinfo(text_sel);
status('Notice that the base addr increased by one line but ' +
      'the limit decreased by 2 lines');
status('This should give you the hint that the limit is ' +
      'relative to the base');
{ fill the descriptor area }
seg_fillword(text_sel, 0, (get_segment_limit(text_sel)+1) div 2,
             makechar(#176, LightMagenta or Brown shl 4));

status('Now let"s get crazy and copy 10 lines of data from ' +
      'the previously saved screen');
{ copy memory from the data segment to screen }
seg_move(get_ds, longint(@text_save), text_sel,
    maxx * bytespercell * 2, maxx * bytespercell * 10);

status('At last freeing the descriptor and restoring the old ' +
      'screen contents..');
status('I hope this little program may give you some hints ' +
      'on working with descriptors');
{ free the descriptor so that it can be used for things }
free_ldt_descriptor(text_sel);
{ restore old state }
seg_move(get_ds, longint(@text_save), dosmemselector,
    linB8000, screensize);
gotoxy(text_oldx, text_oldy);
end.

```

54.19.2 allocate_memory_block

Synopsis: Allocate a block of linear memory.

Declaration: `function allocate_memory_block(size: LongInt) : LongInt`

Visibility: default

Description: Allocates a block of linear memory.

Parameters:

size: Size of requested linear memory block in bytes.

Returned values: blockhandle - the memory handle to this memory block. Linear address of the requested memory.

Remark *warning* According to my DPMI docs this function is not implemented correctly. Normally you should also get a blockhandle to this block after successful operation. This handle can then be used to free the memory block afterwards or use this handle for other purposes. Since the function isn't implemented correctly, and doesn't return a blockhandle, the block can't be deallocated and is hence unusable ! This function doesn't allocate any descriptors for this block, it's the applications responsibility to allocate and initialize for accessing this memory.

Errors: Check the int31error (834) variable.

See also: free_memory_block (841)

54.19.3 copyfromdos

Synopsis: Copy data from DOS to heap.

Declaration: `procedure copyfromdos(var addr; len: LongInt)`

Visibility: default

Description: Copies data from the pre-allocated dos memory transfer buffer to the heap.

Parameters:

addr data to copy to.

len number of bytes to copy to heap.

Notes: Can only be used in conjunction with the dos memory transfer buffer.

Errors: Check the int31error (834) variable.

See also: tb_size (868), transfer_buffer (869), copytodos (838)

54.19.4 copytodos

Synopsis: Copy data from heap to DOS memory.

Declaration: `procedure copytodos(var addr; len: LongInt)`

Visibility: default

Description: Copies data from heap to the pre-allocated dos memory buffer.

Parameters:

addr data to copy from.

len number of bytes to copy to dos memory buffer.

Notes: This function fails if you try to copy more bytes than the transfer buffer is in size. It can only be used in conjunction with the transfer buffer.

Errors: Check the int31error (834) variable.

See also: tb_size (868), transfer_buffer (869), copyfromdos (838)

54.19.5 create_code_segment_alias_descriptor

Synopsis: Create new descriptor from existing descriptor.

Declaration: `function create_code_segment_alias_descriptor(seg: Word) : Word`

Visibility: default

Description: Creates a new descriptor that has the same base and limit as the specified descriptor.

Parameters:

seg Descriptor.

Return values: The data selector (alias).

Notes: In effect, the function returns a copy of the descriptor. The descriptor alias returned by this function will not track changes to the original descriptor. In other words, if an alias is created with this function, and the base or limit of the original segment is then changed, the two descriptors will no longer map the same memory.

Errors: Check the `int31error` (834) variable.

See also: `allocate_ldt_descriptors` (834), `set_segment_limit` (867), `set_segment_base_address` (867)

54.19.6 disable

Synopsis: Disable hardware interrupts.

Declaration: `procedure disable`

Visibility: default

Description: Disables all hardware interrupts by execution a CLI instruction.

Errors: None.

See also: `enable` (841)

54.19.7 dpmi_dosmemfillchar

Synopsis: Fill DOS memory with a character.

Declaration: `procedure dpmi_dosmemfillchar(seg: Word; ofs: Word; count: LongInt; c: Char)`

Visibility: default

Description: `dpmi_dosmemfillchar` fills the DOS memory region indicated by `seg,ofs` with `count` characters `c`.

See also: `dpmi_dosmempu` (840), `dpmi_dosmemget` (840), `dpmi_dosmemmove` (840), `dpmi_dosmemfillword` (840)

54.19.8 dpmi_dosmemfillword

Synopsis: Fill DOS memory with a word value.

Declaration: `procedure dpmi_dosmemfillword(seg: Word; ofs: Word; count: LongInt; w: Word)`

Visibility: default

Description: `dpmi_dosmemfillword` fills the DOS memory region indicated by `seg,ofs` with `count` words `W`.

See also: `dpmi_dosmempout` (840), `dpmi_dosmemget` (840), `dpmi_dosmemfillchar` (839), `dpmi_dosmemmove` (840)

54.19.9 dpmi_dosmemget

Synopsis: Move data from DOS memory to DPMI memory.

Declaration: `procedure dpmi_dosmemget(seg: Word; ofs: Word; var data; count: LongInt)`

Visibility: default

Description: `dpmi_dosmempout` moves `count` bytes of data from the DOS memory location indicated by `seg` and `ofs` to DPMI memory indicated by `data`.

See also: `dpmi_dosmempout` (840), `dpmi_dosmemmove` (840), `dpmi_dosmemfillchar` (839), `dpmi_dosmemfillword` (840)

54.19.10 dpmi_dosmemmove

Synopsis: Move DOS memory.

Declaration: `procedure dpmi_dosmemmove(sseg: Word; sofs: Word; dseg: Word; dofs: Word; count: LongInt)`

Visibility: default

Description: `dpmi_dosmemmove` moves `count` bytes from DOS memory `sseg,sofs` to `dseg,dofs`.

See also: `dpmi_dosmempout` (840), `dpmi_dosmemget` (840), `dpmi_dosmemfillchar` (839), `dpmi_dosmemfillword` (840)

54.19.11 dpmi_dosmempout

Synopsis: Move data from DPMI memory to DOS memory.

Declaration: `procedure dpmi_dosmempout(seg: Word; ofs: Word; var data; count: LongInt)`

Visibility: default

Description: `dpmi_dosmempout` moves `count` bytes of data from `data` to the DOS memory location indicated by `seg` and `ofs`.

See also: `dpmi_dosmemget` (840), `dpmi_dosmemmove` (840), `dpmi_dosmemfillchar` (839), `dpmi_dosmemfillword` (840)

54.19.12 enable

Synopsis: Enable hardware interrupts.

Declaration: `procedure enable`

Visibility: default

Description: Enables all hardware interrupts by executing a STI instruction.

Errors: None.

See also: `disable` (839)

54.19.13 free_ldt_descriptor

Synopsis: Free a descriptor.

Declaration: `function free_ldt_descriptor(d: Word) : Boolean`

Visibility: default

Description: Frees a previously allocated descriptor.

Parameters:

des The descriptor to be freed.

Return value: `True` if successful, `False` otherwise. Notes: After this call this selector is invalid and must not be used for any memory operations anymore. Each descriptor allocated with `allocate_ldt_descriptors` (834) must be freed individually with this function, even if it was previously allocated as a part of a contiguous array of descriptors.

For an example, see `allocate_ldt_descriptors` (834).

Errors: Check the `int31error` (834) variable.

See also: `allocate_ldt_descriptors` (834), `get_next_selector_increment_value` (845)

54.19.14 free_linear_addr_mapping

Synopsis: ? No description available.

Declaration: `function free_linear_addr_mapping(linear_addr: DWord) : Boolean`

Visibility: default

54.19.15 free_memory_block

Synopsis: Free allocated memory block.

Declaration: `function free_memory_block(blockhandle: LongInt) : Boolean`

Visibility: default

Description: Frees a previously allocated memory block.

Parameters:

blockhandle the handle to the memory area to free.

Return value: `True` if successful, `false` otherwise. Notes: Frees memory that was previously allocated with `allocate_memory_block` (837) . This function doesn't free any descriptors mapped to this block, it's the application's responsibility.

Errors: Check `int31error` (834) variable.

See also: `allocate_memory_block` (837)

54.19.16 `free_rm_callback`

Synopsis: Release real mode callback.

Declaration: `function free_rm_callback(var intaddr: tseginfo) : Boolean`

Visibility: `default`

Description: Releases a real mode callback address that was previously allocated with the `get_rm_callback` (847) function.

Parameters:

intaddr real mode address buffer returned by `get_rm_callback` (847) .

Return values: `True` if successful, `False` if not

For an example, see `get_rm_callback` (847).

Errors: Check the `int31error` (834) variable.

See also: `set_rm_interrupt` (867), `get_rm_callback` (847)

54.19.17 `get_cs`

Synopsis: Get CS selector.

Declaration: `function get_cs : Word`

Visibility: `default`

Description: Returns the cs selector.

Return value: The content of the cs segment register.

For an example, see `set_pm_interrupt` (865).

Errors: None.

See also: `get_ds` (843), `get_ss` (853)

54.19.18 `get_descriptor_access_right`

Synopsis: Get descriptor's access rights.

Declaration: `function get_descriptor_access_right(d: Word) : LongInt`

Visibility: `default`

Description: Gets the access rights of a descriptor.

Parameters:

d selector to descriptor.

Return value: Access rights bit field.

Errors: Check the `int31error` (834) variable.

See also: `set_descriptor_access_right` (864)

54.19.19 `get_dpmi_version`

Synopsis: Return DPMI information.

Declaration: `function get_dpmi_version(var version: tdpmiversioninfo) : Boolean`

Visibility: default

Description: `get_dpmi_version` returns version information (Int \$31 Function \$0400) in `Version` and returns `True` if the information was retrieved successfully, `false` if the call failed.

Errors: The call returns `false` if the information could not be retrieved.

See also: `tdpmiversioninfo` (870)

54.19.20 `get_ds`

Synopsis: Get DS Selector.

Declaration: `function get_ds : Word`

Visibility: default

Description: Returns the ds selector.

Return values: The content of the ds segment register.

Errors: None.

See also: `get_cs` (842), `get_ss` (853)

54.19.21 `get_exception_handler`

Synopsis: Return current exception handler.

Declaration: `function get_exception_handler(e: Byte; var intaddr: tseginfo) : Boolean`

Visibility: default

Description: `get_exception_handler` returns the exception handler for exception `E` in `intaddr`. It returns `True` if the call was successful, `False` if not.

See also: `set_exception_handler` (864), `get_pm_exception_handler` (846)

54.19.22 get_linear_addr

Synopsis: Convert physical to linear address.

Declaration: `function get_linear_addr(phys_addr: DWord; size: LongInt) : DWord`

Visibility: default

Description: Converts a physical address into a linear address.

Parameters:

phys_addr physical address of device.

size Size of region to map in bytes.

Return value: Linear address that can be used to access the physical memory. Notes: It's the applications responsibility to allocate and set up a descriptor for access to the memory. This function shouldn't be used to map real mode addresses.

Errors: Check the `int31error` (834) variable.

See also: `allocate_ldt_descriptors` (834), `set_segment_limit` (867), `set_segment_base_address` (867)

54.19.23 get_meminfo

Synopsis: Return information on the available memory.

Declaration: `function get_meminfo(var meminfo: tmeminfo) : Boolean`

Visibility: default

Description: Returns information about the amount of available physical memory, linear address space, and disk space for page swapping.

Parameters:

meminfo buffer to fill memory information into.

Return values: Due to an implementation bug this function always returns `False`, but it always succeeds.

Remark Notes: Only the first field of the returned structure is guaranteed to contain a valid value. Any fields that are not supported by the DPMI host will be set by the host to `-1` (`0FFFFFFFFH`) to indicate that the information is not available. The size of the pages used by the DPMI host can be obtained with the `get_page_size` (846) function.

Errors: Check the `int31error` (834) variable.

See also: `get_page_size` (846)

Listing: `./examples/go32ex/meminfo.pp`

{ Shows how to obtain memory information via get_meminfo();

notice the checks if any of the returned information is invalid (-1)
}

uses

`go32;`

var

```

    meminfo : tmeminfo;

begin
    get_meminfo(meminfo);
    if (int31error <> 0) then begin
        Writeln('Error getting DPML memory information... Halting');
        Writeln('DPML error number : ', int31error);
    end else begin
        with meminfo do begin
            Writeln('Largest available free block : ',
                available_memory div 1024, ' kbytes');
            if (available_pages <> -1) then
                Writeln('Maximum available unlocked pages : ',
                    available_pages);
            if (available_lockable_pages <> -1) then
                Writeln('Maximum lockable available pages : ',
                    available_lockable_pages);
            if (linear_space <> -1) then
                Writeln('Linear address space size : ',
                    linear_space*get_page_size div 1024, ' kbytes');
            if (unlocked_pages <> -1) then
                Writeln('Total number of unlocked pages : ',
                    unlocked_pages);
            if (available_physical_pages <> -1) then
                Writeln('Total number of free pages : ',
                    available_physical_pages);
            if (total_physical_pages <> -1) then
                Writeln('Total number of physical pages : ',
                    total_physical_pages);
            if (free_linear_space <> -1) then
                Writeln('Free linear address space : ',
                    free_linear_space*get_page_size div 1024,
                    ' kbytes');
            if (max_pages_in_paging_file <> -1) then
                Writeln('Maximum size of paging file : ',
                    max_pages_in_paging_file*get_page_size div 1024,
                    ' kbytes');
        end;
    end;
end.

```

54.19.24 get_next_selector_increment_value

Synopsis: Return selector increment value.

Declaration: `function get_next_selector_increment_value : Word`

Visibility: default

Description: Returns the selector increment value when allocating multiple subsequent descriptors via `allocate_ldt_descriptors` (834).

Return value: Selector increment value.

Remark Notes: Because `allocate_ldt_descriptors` (834) only returns the selector for the first descriptor and so the value returned by this function can be used to calculate the selectors for subsequent descriptors in the array.

Errors: Check the `int31error` (834) variable.

See also: [allocate_ldt_descriptors \(834\)](#), [free_ldt_descriptor \(841\)](#)

54.19.25 `get_page_attributes`

Synopsis: ? No description available.

Declaration: `function get_page_attributes(handle: DWord; offset: DWord;
pagecount: DWord; buf: pointer) : Boolean`

Visibility: default

54.19.26 `get_page_size`

Synopsis: Return the page size.

Declaration: `function get_page_size : LongInt`

Visibility: default

Description: Returns the size of a single memory page.

Return value: Size of a single page in bytes.

Remark The returned size is typically 4096 bytes.

For an example, see [get_meminfo \(844\)](#).

Errors: Check the [int31error \(834\)](#) variable.

See also: [get_meminfo \(844\)](#)

54.19.27 `get_pm_exception_handler`

Synopsis: Get protected mode exception handler.

Declaration: `function get_pm_exception_handler(e: Byte; var intaddr: tseginfo)
: Boolean`

Visibility: default

Description: `get_pm_exception_handler` returns the protected mode exception handler for exception `E` in `intaddr`. It returns `True` if the call was successful, `False` if not.

See also: [get_exception_handler \(843\)](#), [set_pm_exception_handler \(865\)](#)

54.19.28 `get_pm_interrupt`

Synopsis: Return protected mode interrupt handler.

Declaration: `function get_pm_interrupt(vector: Byte; var intaddr: tseginfo) : Boolean`

Visibility: default

Description: Returns the address of a current protected mode interrupt handler.

Parameters:

vector interrupt handler number you want the address to.

intaddr buffer to store address.

Return values: `True` if successful, `False` if not.

Remark The returned address is a protected mode selector:offset address.

For an example, see `set_pm_interrupt` (865).

Errors: Check the `int31error` (834) variable.

See also: `set_pm_interrupt` (865), `set_rm_interrupt` (867), `get_rm_interrupt` (851)

54.19.29 `get_rm_callback`

Synopsis: Return real mode callback.

Declaration: `function get_rm_callback(pm_func: pointer; const reg: trealregs;
var rmcb: tseginfo) : Boolean`

Visibility: default

Description: Returns a unique real mode `segment:offset` address, known as a "real mode callback," that will transfer control from real mode to a protected mode procedure.

Parameters:

pm_func pointer to the protected mode callback function.

reg supplied registers structure.

rmcb buffer to real mode address of callback function.

Return values: `True` if successful, otherwise `False`.

Remark Callback addresses obtained with this function can be passed by a protected mode program for example to an interrupt handler, device driver, or TSR, so that the real mode program can call procedures within the protected mode program or notify the protected mode program of an event. The contents of the supplied `regs` structure is not valid after function call, but only at the time of the actual callback.

Errors: Check the `int31error` (834) variable.

See also: `free_rm_callback` (842)

Listing: `./examples/go32ex/callback.pp`

{ This program tries to give an example how to install a callback procedure with the help of the GO32 unit.

It installs a callback which is supplied by any Microsoft compatible mouse driver; at a specified mouse action this routine is called. This callback must provide the services explained in the docs. The main callback has to be in assembly, because it isn't possible to do these services with pascal alone. But is written as general as possible to provide maximum re-usability for other applications and hence it simply calls a normal pascal user procedure in addition to some initialization and callback service code, so you don't need to hassle around with it too much.

Notes to this user procedure :

**) it should not last too long to execute it*

**) ALL data and code touched in this proc MUST be locked BEFORE it is called the first time*

Used software interrupt calls (rough descriptions, only what's used):

Int 33h 0000h – Microsoft Mouse driver : Reset mouse

Input : AX = 0000h

Return : AX = FFFFh if successful

BX = number of buttons (if FFFFh then mouse has 2 buttons)

Int 33h 0001h – Microsoft Mouse driver : Show mouse cursor

Input : AX = 0001h

Return : Mouse cursor shown on screen

Int 33h 0002h – Microsoft mouse driver : Hide mouse cursor

Input : AX = 0002h

Return : Hides mouse cursor again

Int 33h 000Ch – Microsoft mouse driver : Install user callback

Input : AX = 000Ch

CX = bit mask which tells the mouse driver at which actions the callback should be called, i.e. if button pressed, mouse moved etc.

(In this example it's set to 7Fh so that the callback is called on every action)

ES:EDX = pointer to callback procedure to call

Note : The registers structure supplied to the callback contains valid mouse data when the handler is called.

BX = button state information

CX = mouse X coordinates

DX = mouse Y coordinates

For more detailed information consult any mouse reference or interrupt list.

}

{ \$CALLING REGISTER }

{ \$ASMMODE ATT }

{ \$MODE FPC }

uses

*crt,
go32;*

const

{ the mouse interrupt number }

mouseint = \$33;

var

{ supplied register structure to the callback }

*mouse_regs : trealregs; **external name** '___v2prt0_rmcb_regs';*

{ real mode 48 bit pointer to the callback }

mouse_seginfo : tseginfo;

var

{ number of mouse buttons }

mouse_numbuttons : longint;

{ bit mask for the action which triggered the callback }

mouse_action : word;

{ current mouse x and y coordinates }

mouse_x, mouse_y : Word;

```

    { button state }
    mouse_b : Word;

    { is an additional user procedure installed }
    userproc_installed : Longbool;
    { length of additional user procedure }
    userproc_length : Longint;
    { pointer to user proc }
    userproc_proc : pointer;

{ callback control handler, calls a user procedure if installed }

{ callback control handler, calls a user procedure if installed }
procedure callback_handler; assembler;
asm
    pushw %ds
    pushl %eax
    movw %es, %ax
    movw %ax, %ds

    { give control to user procedure if installed }
    cmpl $0, USERPROC_INSTALLED
    je .LNoCallback
    pushal
    movw DOSmemSELECTOR, %ax
    movw %ax, %fs { set fs for FPC }
    call *USERPROC_PROC
    popal
.LNoCallback:

    popl %eax
    popw %ds

    pushl %eax
    movl (%esi), %eax
    movl %eax, %es: 42(%edi) { adjust stack }
    addw $4, %es:46(%edi)
    popl %eax
    iret
end;
{ This dummy is used to obtain the length of the callback control
function. It has to be right after the callback_handler() function.
}
procedure mouse_dummy; begin end;

{ This is the supplied user procedure. In this case we simply
transform the virtual 640x200 mouse coordinate system to a 80x25
text mode coordinate system }
procedure textuserproc;
begin
    { the mouse_regs record contains the real mode registers now }
    mouse_b := mouse_regs.bx;
    mouse_x := (mouse_regs.cx shr 3) + 1;
    mouse_y := (mouse_regs.dx shr 3) + 1;
end;

{ Description : Installs the mouse callback control handler and
handles all necessary mouse related initialization.

```

```

    Input : userproc – pointer to a user procedure, nil if none
           userproclen – length of user procedure
}
procedure install_mouse(userproc : pointer; userproclen : longint);
var r : trealregs;
begin
    { mouse driver reset }
    r.eax := $0; realintr(mouseint, r);
    if (r.eax <> $FFFF) then begin
        Writeln('No Microsoft compatible mouse found');
        Writeln('A Microsoft compatible mouse driver is necessary ',
            'to run this example');
        halt;
    end;
    { obtain number of mouse buttons }
    if (r.bx = $ffff) then mouse_numbuttons := 2
    else mouse_numbuttons := r.bx;
    Writeln(mouse_numbuttons, ' button Microsoft compatible mouse ',
        ' found. ');
    { check for additional user procedure, and install it if
    available }
    if (userproc <> nil) then begin
        userproc_proc := userproc;
        userproc_installed := true;
        userproc_length := userproclen;
        { lock code for user procedure }
        lock_code(userproc_proc, userproc_length);
    end else begin
        { clear variables }
        userproc_proc := nil;
        userproc_length := 0;
        userproc_installed := false;
    end;
    { lock code & data which is touched in the callback handler }
    lock_data(mouse_x, sizeof(mouse_x));
    lock_data(mouse_y, sizeof(mouse_y));
    lock_data(mouse_b, sizeof(mouse_b));
    lock_data(mouse_action, sizeof(mouse_action));

    lock_data(userproc_installed, sizeof(userproc_installed));
    lock_data(userproc_proc, sizeof(userproc_proc));

    lock_data(mouse_regs, sizeof(mouse_regs));
    lock_data(mouse_seginf, sizeof(mouse_seginf));
    lock_code(@callback_handler,
        longint(@mouse_dummy)–longint(@callback_handler));
    { allocate callback (supply registers structure) }
    get_rm_callback(@callback_handler, mouse_regs, mouse_seginf);
    { install callback }
    r.eax := $0c; r.ecx := $7f;
    r.edx := longint(mouse_seginf.offset);
    r.es := mouse_seginf.segment;
    realintr(mouseint, r);
    { show mouse cursor }
    r.eax := $01;
    realintr(mouseint, r);
end;

```

```

procedure remove_mouse;
var
    r : trealregs;
begin
    { hide mouse cursor }
    r.eax := $02; realintr(mouseint, r);
    { remove callback handler }
    r.eax := $0c; r.ecx := 0; r.edx := 0; r.es := 0;
    realintr(mouseint, r);
    { free callback }
    free_rm_callback(mouse_seginfo);
    { check if additional userproc is installed, and clean up if
      needed }
    if (userproc_installed) then begin
        unlock_code(userproc_proc, userproc_length);
        userproc_proc := nil;
        userproc_length := 0;
        userproc_installed := false;
    end;
    { unlock used code & data }
    unlock_data(mouse_x, sizeof(mouse_x));
    unlock_data(mouse_y, sizeof(mouse_y));
    unlock_data(mouse_b, sizeof(mouse_b));
    unlock_data(mouse_action, sizeof(mouse_action));

    unlock_data(userproc_proc, sizeof(userproc_proc));
    unlock_data(userproc_installed, sizeof(userproc_installed));

    unlock_data(mouse_regs, sizeof(mouse_regs));
    unlock_data(mouse_seginfo, sizeof(mouse_seginfo));
    unlock_code(@callback_handler,
        longint(@mouse_dummy)-longint(@callback_handler));
    fillchar(mouse_seginfo, sizeof(mouse_seginfo), 0);
end;

begin
    install_mouse(@textuserproc, 400);
    Writeln('Press any key to exit...');
    while (not keypressed) do begin
        { write mouse state info }
        gotoxy(1, wherey);
        write('MouseX : ', mouse_x:2, ' MouseY : ', mouse_y:2,
            ' Buttons : ', mouse_b:2);
    end;
    remove_mouse;
end.

```

54.19.30 get_rm_interrupt

Synopsis: Get real mode interrupt vector.

Declaration: `function get_rm_interrupt(vector: Byte; var intaddr: tseginfo) : Boolean`

Visibility: default

Description: Returns the contents of the current machine's real mode interrupt vector for the specified interrupt.

Parameters:

vector interrupt vector number.

intaddr buffer to store real mode `segment:offset` address.

Return values: `True` if successful, `False` otherwise.

Remark The returned address is a real mode segment address, which isn't valid in protected mode.

Errors: Check the `int31error` (834) variable.

See also: `set_rm_interrupt` (867), `set_pm_interrupt` (865), `get_pm_interrupt` (846)

54.19.31 `get_run_mode`

Synopsis: Return current run mode.

Declaration: `function get_run_mode : Word`

Visibility: `default`

Description: Returns the current mode your application runs with.

Return values: One of the constants used by this function.

Errors: None.

See also: `get_run_mode` (852)

Listing: `./examples/go32ex/getrunmd.pp`

{ Simply write a message according to the current environment }

uses

`go32;`

begin

*{ depending on the detected environment we simply write
another message Note: in go32v2 this will always be `rm_dpml`. }*

case (`get_run_mode`) **of**

`rm_unknown :`

Writeln('Unknown environment found');

`rm_raw :`

Writeln('You are currently running in raw mode ',
'(without HIMEM)');

`rm_xms :`

Writeln('You are currently using HIMEM.SYS only');

`rm_vcpi :`

Writeln('VCPI server detected. You're using HIMEM and ',
'EMM386');

`rm_dpml :`

Writeln('DPMI detected. You're using a DPMI host like ',
'a windows DOS box or CWSDPMI');

end;

end.

54.19.32 get_segment_base_address

Synopsis: Return base address from descriptor table.

Declaration: `function get_segment_base_address(d: Word) : DWord`

Visibility: default

Description: Returns the 32-bit linear base address from the descriptor table for the specified segment.

Parameters:

d selector of the descriptor you want the base address of.

Return values: Linear base address of specified descriptor.

For an example, see `allocate_ldt_descriptors` (834).

Errors: Check the `int31error` (834) variable.

See also: `allocate_ldt_descriptors` (834), `set_segment_base_address` (867), `allocate_ldt_descriptors` (834), `set_segment_limit` (867), `get_segment_limit` (853)

54.19.33 get_segment_limit

Synopsis: Return segment limits from descriptor.

Declaration: `function get_segment_limit(d: Word) : DWord`

Visibility: default

Description: Returns a descriptors segment limit.

Parameters:

d selector.

Return value: Limit of the descriptor in bytes.

Errors: Returns zero if descriptor is invalid.

See also: `allocate_ldt_descriptors` (834), `set_segment_limit` (867), `set_segment_base_address` (867), `get_segment_base_address` (853)

54.19.34 get_ss

Synopsis: Return SS selector.

Declaration: `function get_ss : Word`

Visibility: default

Description: Returns the ss selector.

Return values: The content of the ss segment register.

Errors: None.

See also: `get_ds` (843), `get_cs` (842)

54.19.35 global_dos_alloc

Synopsis: Allocate DOS real mode memory.

Declaration: `function global_dos_alloc(bytes: LongInt) : LongInt`

Visibility: default

Description: Allocates a block of dos real mode memory.

Parameters:

bytes size of requested real mode memory.

Return values: The low word of the returned value contains the selector to the allocated dos memory block, the high word the corresponding real mode segment value. The offset value is always zero. This function allocates memory from dos memory pool, i.e. memory below the 1 MB boundary that is controlled by dos. Such memory blocks are typically used to exchange data with real mode programs, TSRs, or device drivers. The function returns both the real mode segment base address of the block and one descriptor that can be used by protected mode applications to access the block. This function should only be used for temporary buffers to get real mode information (e.g. interrupts that need a data structure in ES:(E)DI), because every single block needs a unique selector. The returned selector should only be freed by a `global_dos_free` (856) call.

Errors: Check the `int31error` (834) variable.

See also: `global_dos_free` (856)

Listing: `./examples/go32ex/buffer.pp`

{ This program demonstrates the usage of DOS real mode memory by executing a software interrupt which needs a buffer to store data into. Because these interrupts are real mode funcs, the buffer must be located in real mode memory space (first MB of memory). Such memory can only be allocated by the `global_dos_alloc()` and `global_dos_free()` functions of the GO32 unit.

In more detail this program tries to detect a VESA 2.0 BIOS extension of your graphics card and outputs its version.

Here's the necessary interrupt call description:

Int 10h 4f00h : VESA BIOS extension installation check

Input : AX = 4F00h

ES:DI = pointer to 512 byte information buffer

Output : AX = 004Fh if successful

ES:DI = pointer to filled buffer

Buffer structure : (relevant to this example)

must be 'VESA' in the first 4 chars of the buffer to be valid VBE version in the next word

Note : to request VBE 2.0 information, the first 4 bytes of the buffer must contain 'VBE2' prior to the interrupt call.

(this makes the problem a bit tougher; we first have to copy the buffer with the 'VBE2' id to dos memory...)

}

uses

go32;

*{The following 2 functions are wrappers to the GO32
global_dos_alloc() and global_dos_free() functions to simplify their
usage }*

{ Function : dosalloc }

{ Input : size of a real mode location }

{ Output : selector and segment of a real mode location }

procedure dosalloc(**var** selector : word;
 var segment : word; size : longint);

var

res : longint;

begin

{ try to allocate real mode memory }

res := global_dos_alloc(size);

*{ the lower 16 bits of the result contain the selector to the
allocated memory block }*

selector := word(res);

*{ the upper 16 bits contain the real mode segment address of
this block; the offset is always 0, so we don't need to return
this }*

segment := word(res shr 16);

end;

{ Function : dosfree }

{ Input : selector of a real mode block }

{ Output : none }

*{ Description : de-allocates a previously allocated real mode
memory }*

procedure dosfree(selector : word);

begin

{ call the GO32 function with the selector }

global_dos_free(selector);

end;

type

VBEInfoBuf = **packed record**

{ contains 'VESA' if successful }

Signature : **array**[0..3] **of** char;

Version : Word;

{ pad to 512 bytes length }

reserved : **array**[0..505] **of** byte;

end;

var

{ selector to our real mode buffer }

selector,

{ real mode segment address of buffer }

segment : Word;

{ register structure to issue a software interrupt }

r : trealregs;

infobuf : VBEInfoBuf;

begin

{ first we reset the registers and infobuf variable }

```

fillchar(r, sizeof(r), 0);
fillchar(infobuf, sizeof(VBEInfoBuf), 0);
{ allocate real mode memory }
dosalloc(selector, segment, sizeof(VBEInfoBuf));
{ check if an error occurred during allocation }
if (int31error<>0) then begin
    Writeln('Error while allocating real mode memory, halting');
    halt;
end;
{ request VBE 2.0 information, fill out information buffer }
infobuf.Signature := 'VBE2';
{ copy buffer to the allocated real mode memory }
dosmemput(segment, 0, infobuf, sizeof(infobuf));
{ issue the interrupt; remember : DI = 0 }
r.ax := $4f00; r.es := segment;
realintr($10, r);
{ copy buffer to our infobuf variable again }
dosmemget(segment, 0, infobuf, sizeof(infobuf));
{ free allocated real mode memory, because we don't need it anymore }
dosfree(selector);
{ check if interrupt call was successful }
if (r.ax <> $4f) then begin
    { write message and exit, because the infobuf doesn't contain any useful data we could tell the user }
    Writeln('VBE BIOS extension not available, function call ',
        'failed');
    halt;
end;
{ check if buffer is valid }
if (infobuf.signature[0] = 'V') and
    (infobuf.signature[1] = 'E') and
    (infobuf.signature[2] = 'S') and
    (infobuf.signature[3] = 'A') then begin
    Writeln('VBE version ', hi(infobuf.version), '.',
        lo(infobuf.version), ' detected');
end;
end.

```

54.19.36 global_dos_free

Synopsis: Free DOS memory block.

Declaration: `function global_dos_free(selector: Word) : Boolean`

Visibility: default

Description: Frees a previously allocated dos memory block.

Parameters:

selector selector to the dos memory block.

Return value: `True` if successful, `False` otherwise.

Remark The descriptor allocated for the memory block is automatically freed and hence invalid for further use. This function should only be used for memory allocated by `global_dos_alloc` (854).

For an example, see `global_dos_alloc` (854).

Errors: Check the `int31error` (834) variable.

See also: `global_dos_alloc` (854)

54.19.37 `inportb`

Synopsis: Read byte from I/O port.

Declaration: `function inportb(port: Word) : Byte`

Visibility: default

Description: Reads 1 byte from the selected I/O port.

Parameters:

port the I/O port number which is read.

Return values: Current I/O port value.

Errors: None.

See also: `outportb` (859), `inportw` (857), `inportl` (857)

54.19.38 `inportl`

Synopsis: Read longint from I/O port.

Declaration: `function inportl(port: Word) : LongInt`

Visibility: default

Description: Reads 1 longint from the selected I/O port.

Parameters:

port the I/O port number which is read.

Return values: Current I/O port value.

Errors: None.

See also: `outportb` (859), `inportb` (857), `inportw` (857)

54.19.39 `inportw`

Synopsis: Read word from I/O port.

Declaration: `function inportw(port: Word) : Word`

Visibility: default

Description: Reads 1 word from the selected I/O port.

Parameters:

port the I/O port number which is read.

Return values: Current I/O port value.

Errors: None.

See also: `outportw` (860), `inportb` (857), `inportl` (857)

54.19.40 lock_code

Synopsis: Lock code memory range.

Declaration: `function lock_code(functionaddr: pointer; size: LongInt) : Boolean`

Visibility: default

Description: Locks a memory range which is in the code segment selector.

Parameters:

functionaddr address of the function to be locked.

size size in bytes to be locked.

Return values: `True` if successful, `False` otherwise.

For an example, see `get_rm_callback` (847).

Errors: Check the `int31error` (834) variable.

See also: `lock_linear_region` (858), `lock_data` (858), `unlock_linear_region` (870), `unlock_data` (869), `unlock_code` (869)

54.19.41 lock_data

Synopsis: Lock data memory range.

Declaration: `function lock_data(var data; size: LongInt) : Boolean`

Visibility: default

Description: Locks a memory range which resides in the data segment selector.

Parameters:

data address of data to be locked.

size length of data to be locked.

Return values: `True` if successful, `False` otherwise.

For an example, see `get_rm_callback` (847).

Errors: Check the `int31error` (834) variable.

See also: `lock_linear_region` (858), `lock_code` (858), `unlock_linear_region` (870), `unlock_data` (869), `unlock_code` (869)

54.19.42 lock_linear_region

Synopsis: Lock linear memory region.

Declaration: `function lock_linear_region(linearaddr: LongInt; size: LongInt)
: Boolean`

Visibility: default

Description: Locks a memory region to prevent swapping of it.

Parameters:

linearaddr the linear address of the memory are to be locked.

size size in bytes to be locked.

Return value: True if successful, False otherwise.

Errors: Check the `int31error` (834) variable.

See also: `lock_data` (858), `lock_code` (858), `unlock_linear_region` (870), `unlock_data` (869), `unlock_code` (869)

54.19.43 map_device_in_memory_block

Synopsis: Map a device into program's memory space.

Declaration: `function map_device_in_memory_block(handle: DWord; offset: DWord; pagecount: DWord; device: DWord) : Boolean`

Visibility: default

Description: `map_device_in_memory_block` allows to map a device in memory. This function is a direct call of the extender. For more information about it's arguments, see the extender documentation.

54.19.44 outportb

Synopsis: Write byte to I/O port.

Declaration: `procedure outportb(port: Word; data: Byte)`

Visibility: default

Description: Sends 1 byte of data to the specified I/O port.

Parameters:

port the I/O port number to send data to.

data value sent to I/O port.

Return values: None.

Errors: None.

See also: `inportb` (857), `outportl` (860), `outportw` (860)

Listing: `./examples/go32ex/outport.pp`

{ This example demonstrates the use of the outport functions.

It simply turns the PC's internal speaker on for 50 ms and off again

uses

`crt,`
`go32;`

begin

{ turn on speaker }
`outportb($61, $ff);`
{ wait a little bit }
`delay(50);`
{ turn it off again }
`outportb($61, $0);`

end.

54.19.45 outportl

Synopsis: Write longint to I/O port.

Declaration: `procedure outportl(port: Word; data: LongInt)`

Visibility: default

Description: Sends 1 longint of data to the specified I/O port.

Parameters:

port the I/O port number to send data to.

data value sent to I/O port.

Return values: None.

For an example, see `outportb` (859).

Errors: None.

See also: `inportl` (857), `outportw` (860), `outportb` (859)

54.19.46 outportw

Synopsis: Write word to I/O port.

Declaration: `procedure outportw(port: Word; data: Word)`

Visibility: default

Description: Sends 1 word of data to the specified I/O port.

Parameters:

port the I/O port number to send data to.

data value sent to I/O port.

Return values: None.

For an example, see `outportb` (859).

Errors: None.

See also: `inportw` (857), `outportl` (860), `outportb` (859)

54.19.47 realintr

Synopsis: Simulate interrupt.

Declaration: `function realintr(intnr: Word; var regs: trealregs) : Boolean`

Visibility: default

Description: Simulates an interrupt in real mode.

Parameters:

intnr interrupt number to issue in real mode.

regs registers data structure.

Return values: The supplied registers data structure contains the values that were returned by the real mode interrupt. `True` if successful, `False` if not.

Remark The function transfers control to the address specified by the real mode interrupt vector of `intr`. The real mode handler must return by executing an `IRET`.

Errors: Check the `int31error` (834) variable.

Listing: `./examples/go32ex/flags.pp`

{ This example demonstrates the use of the flag constants in conjunction with an interrupt call

In detail it checks if APM (advanced power management) is available.

Int 15h 5300h – APM specification : Installation check

Input : AX = 5300h

BX = device id of system BIOS (= 0000h)

Return : Carry clear if successful

AH = major version (BCD)

AL = minor version (BCD)

}

uses

`go32;`

var

`r : trealregs;`

begin

{ set register values and issue real mode interrupt call }

`r.ax := $5300;`

`r.bx := 0;`

`realintr($15, r);`

{ check if carry clear and write a suited message }

if `((r.flags and carryflag)=0)` **then begin**

`Writeln('APM v', (r.ah and $f), ' ',`

`(r.al shr 4), (r.al and $f), ' detected');`

end else

`Writeln('APM not present');`

end.

54.19.48 request_linear_region

Synopsis: Request linear address region.

Declaration: `function request_linear_region(linearaddr: LongInt; size: LongInt;
var blockhandle: LongInt) : Boolean`

Visibility: default

Description: `request_linear_region` requests a linear range of addresses of size `Size`, starting at `linearaddr`. If successful, `True` is returned, and a handle to the address region is returned in `blockhandle`.

Errors: On error, `False` is returned.

54.19.49 segment_to_descriptor

Synopsis: Map segment address to descriptor.

Declaration: `function segment_to_descriptor(seg: Word) : Word`

Visibility: default

Description: Maps a real mode segment (paragraph) address onto an descriptor that can be used by a protected mode program to access the same memory.

Parameters:

seg the real mode segment you want the descriptor to.

Return values: Descriptor to real mode segment address.

Remark The returned descriptors limit will be set to 64 kB. Multiple calls to this function with the same segment address will return the same selector. Descriptors created by this function can never be modified or freed. Programs which need to examine various real mode addresses using the same selector should use the function `allocate_ldt_descriptors` (834) and change the base address as necessary.

For an example, see `seg_fillchar` (862).

Errors: Check the `int31error` (834) variable.

See also: `allocate_ldt_descriptors` (834), `free_ldt_descriptor` (841), `set_segment_base_address` (867)

54.19.50 seg_fillchar

Synopsis: Fill segment with byte value.

Declaration: `procedure seg_fillchar(seg: Word; ofs: LongInt; count: LongInt; c: Char)`

Visibility: default

Description: Sets a memory area to a specific value.

Parameters:

seg selector to memory area.

ofs offset to memory.

count number of bytes to set.

c byte data which is set.

Return values: None.

Notes: No range check is done in any way.

Errors: None.

See also: `seg_move` (864), `seg_fillword` (863), `dosmemfillchar` (830), `dosmemfillword` (830), `dosmemget` (830), `dosmemput` (831), `dosmemmove` (831)

Listing: `./examples/go32ex/vgasel.pp`

{This example demonstrates the use of the segment_to_descriptor() function.

It switches to VGA mode 13h (320x200x256 color), creates a selector to the memory (based at \$A000:0000), clears this memory with color 15 (white) and waits until the enter key is pressed }

uses go32;

var

vgasel : Word;
r : trealregs;

begin

{ set VGA mode 13h }
r.eax := \$13; realintr(\$10, r);
{ allocate descriptor to VGA memory quickly; it could be done with allocate_ldt_descriptors() too, but we would have to initialize it by ourselves... unlike segment_to_descriptor() which automatically sets the limit and the base address correctly }
vgasel := segment_to_descriptor(\$A000);
{ simply fill the screen memory with color 15 }
seg_fillchar(vgasel, 0, 64000, #15);
{ wait for a return press }
readln;
{ back to text mode }
r.eax := \$3; realintr(\$10, r);
{ don't deallocate vgasel, that can't be done }

end.

54.19.51 seg_fillword

Synopsis: Fill segment with word value.

Declaration: `procedure seg_fillword(seg: Word; ofs: LongInt; count: LongInt; w: Word)`

Visibility: default

Description: Sets a memory area to a specific value.

Parameters:

seg selector to memory area.

ofs offset to memory.

count number of words to set.

w word data which is set.

Return values: None.

Notes: No range check is done in any way.

For an example, see `allocate_ldt_descriptors` (834).

Errors: None.

See also: `seg_move` (864), `seg_fillchar` (862), `dosmemfillchar` (830), `dosmemfillword` (830), `dosmemget` (830), `dosmemput` (831), `dosmemmove` (831)

54.19.52 seg_move

Synopsis: Move data between 2 locations.

Declaration: `procedure seg_move(sseg: Word; source: LongInt; dseg: Word;
dest: LongInt; count: LongInt)`

Visibility: default

Description: Copies data between two memory locations.

Parameters:

sseg source selector.

source source offset.

dseg destination selector.

dest destination offset.

count size in bytes to copy.

Return values: None.

Remark Overlapping is only checked if the source selector is equal to the destination selector. No range check is done.

For an example, see `allocate_ldt_descriptors` (834).

Errors: None.

See also: `seg_fillchar` (862), `seg_fillword` (863), `dosmemfillchar` (830), `dosmemfillword` (830), `dosmemget` (830), `dosmemput` (831), `dosmemmove` (831)

54.19.53 set_descriptor_access_right

Synopsis: Set access rights to memory descriptor.

Declaration: `function set_descriptor_access_right(d: Word; w: Word) : Boolean`

Visibility: default

Description: `set_descriptor_access_right` sets the access rights for descriptor `d` to `w`

54.19.54 set_exception_handler

Synopsis: Set exception handler.

Declaration: `function set_exception_handler(e: Byte; const intaddr: tseginfo)
: Boolean`

Visibility: default

Description: `set_exception_handler` sets the exception handler for exception `E` to `intaddr`. It returns `True` if the call was successful, `False` if not.

See also: `get_exception_handler` (843), `set_pm_exception_handler` (865)

54.19.55 set_page_attributes

Synopsis: ? No description available.

Declaration: `function set_page_attributes(handle: DWord; offset: DWord;
pagecount: DWord; buf: pointer) : Boolean`

Visibility: default

54.19.56 set_pm_exception_handler

Synopsis: Set protected mode exception handler.

Declaration: `function set_pm_exception_handler(e: Byte; const intaddr: tseginfo)
: Boolean`

Visibility: default

Description: `set_pm_exception_handler` sets the protected mode exception handler for exception E to `intaddr`. It returns `True` if the call was successful, `False` if not.

See also: `set_exception_handler` (864), `get_pm_exception_handler` (846)

54.19.57 set_pm_interrupt

Synopsis: Set protected mode interrupt handler.

Declaration: `function set_pm_interrupt(vector: Byte; const intaddr: tseginfo)
: Boolean`

Visibility: default

Description: Sets the address of the protected mode handler for an interrupt.

Parameters:

vector number of protected mode interrupt to set.

intaddr selector:offset address to the interrupt vector.

Return values: `True` if successful, `False` otherwise.

Remark The address supplied must be a valid `selector:offset` protected mode address.

Errors: Check the `int31error` (834) variable.

See also: `get_pm_interrupt` (846), `set_rm_interrupt` (867), `get_rm_interrupt` (851)

Listing: `./examples/go32ex/intpm.pp`

*{ This example shows how to redirect a software interrupt by
changing the protected mode handler of the DPML host.*

*In more detail it hooks interrupt 1Ch which is called every
time the timer interrupt (int 08) is executed. This is the
preferred way to hook the timer, because int 1Ch is a software
interrupt which doesn't need so much initialization stuff
compared to hooking a hardware interrupt.*
}

uses

```

    crt,
    go32;

const
    { interrupt number we want to hook }
    int1c = $1c;

var
    { 48 bit pointer to old interrupt handler }
    oldint1c : tseinfo;
    { 48 bit pointer to new interrupt handler }
    newint1c : tseinfo;

    { increased every time the interrupt is called }
    int1c_counter : Longint;

    { the current data selector }
    int1c_ds : Word; external name '___v2prt0_ds_alias';

{ the actual handler code }
procedure int1c_handler; assembler;
asm
    cli
    { save all registers }
    pushw %ds
    pushw %ax
    { prepare segment registers for FPC procedure }
    movw %cs:int1c_ds, %ax
    movw %ax, %ds
    { simply increase the counter by one }
    incl int1c_counter
    { restore registers }
    popw %ax
    popw %ds
    sti
    iret
end;

var i : Longint;

begin
    { insert right handler data into new handler variable }
    newint1c.offset := @int1c_handler;
    newint1c.segment := get_cs;
    { get the old handler }
    get_pm_interrupt(int1c, oldint1c);
    Writeln('--- Press any key to exit ---');
    { set new handler }
    set_pm_interrupt(int1c, newint1c);
    { write the number of interrupts occurred }
    while (not keypressed) do begin
        gotoxy(1, wherey);
        write('Number of interrupts occurred : ', int1c_counter);
    end;
    { restore old handler }
    set_pm_interrupt(int1c, oldint1c);
end.

```

54.19.58 set_rm_interrupt

Synopsis: Set real mode interrupt handler.

Declaration: `function set_rm_interrupt(vector: Byte; const intaddr: tseginfo)
: Boolean`

Visibility: default

Description: Sets a real mode interrupt handler.

Parameters:

vector the interrupt vector number to set.

intaddr address of new interrupt vector.

Return values: True if successful, otherwise False.

Remark The address supplied MUST be a real mode segment address, not a `selector:offset` address. So the interrupt handler must either reside in dos memory (below 1 Mb boundary) or the application must allocate a real mode callback address with `get_rm_callback` (847).

Errors: Check the `int31error` (834) variable.

See also: `get_rm_interrupt` (851), `set_pm_interrupt` (865), `get_pm_interrupt` (846), `get_rm_callback` (847)

54.19.59 set_segment_base_address

Synopsis: Set descriptor's base address.

Declaration: `function set_segment_base_address(d: Word; s: DWord) : Boolean`

Visibility: default

Description: Sets the 32-bit linear base address of a descriptor.

Parameters:

d selector.

s new base address of the descriptor.

Errors: Check the `int31error` (834) variable.

See also: `allocate_ldt_descriptors` (834), `get_segment_base_address` (853), `allocate_ldt_descriptors` (834), `set_segment_limit` (867), `get_segment_base_address` (853), `get_segment_limit` (853)

54.19.60 set_segment_limit

Synopsis: Set descriptor limit.

Declaration: `function set_segment_limit(d: Word; s: DWord) : Boolean`

Visibility: default

Description: Sets the limit of a descriptor.

Parameters:

d selector.

s new limit of the descriptor.

Return values: Returns `True` if successful, else `False`.

Remark The new limit specified must be the byte length of the segment - 1. Segment limits bigger than or equal to 1MB must be page aligned, they must have the lower 12 bits set.

For an example, see `allocate_ldt_descriptors` (834).

Errors: Check the `int31error` (834) variable.

See also: `allocate_ldt_descriptors` (834), `set_segment_base_address` (867), `get_segment_limit` (853), `set_segment_limit` (867)

54.19.61 `tb_offset`

Synopsis: Return DOS transfer buffer offset.

Declaration: `function tb_offset : LongInt`

Visibility: default

Description: `tb_offset` returns the DOS transfer buffer segment.

See also: `transfer_buffer` (869), `tb_segment` (868), `tb_size` (868)

54.19.62 `tb_segment`

Synopsis: Return DOS transfer buffer segment.

Declaration: `function tb_segment : LongInt`

Visibility: default

Description: `tb_segment` returns the DOS transfer buffer segment.

See also: `transfer_buffer` (869), `tb_offset` (868), `tb_size` (868)

54.19.63 `tb_size`

Synopsis: Return DOS transfer memory buffer size.

Declaration: `function tb_size : LongInt`

Visibility: default

Description: Returns the size of the pre-allocated dos memory buffer.

Return values: The size of the pre-allocated dos memory buffer. This block always seems to be 16k in size, but don't rely on this.

Errors: None.

See also: `transfer_buffer` (869), `copyfromdos` (838), `copytodos` (838)

54.19.64 transfer_buffer

Synopsis: Return offset of DOS transfer buffer.

Declaration: `function transfer_buffer : LongInt`

Visibility: default

Description: `transfer_buffer` returns the offset of the transfer buffer.

Errors: None.

See also: `tb_size` (868)

54.19.65 unlock_code

Synopsis: Unlock code segment.

Declaration: `function unlock_code(functionaddr: pointer; size: LongInt) : Boolean`

Visibility: default

Description: Unlocks a memory range which resides in the code segment selector.

Parameters:

functionaddr address of function to be unlocked.

size size bytes to be unlocked.

Return value: `True` if successful, `False` otherwise.

For an example, see `get_rm_callback` (847).

Errors: Check the `int31error` (834) variable.

See also: `unlock_linear_region` (870), `unlock_data` (869), `lock_linear_region` (858), `lock_data` (858), `lock_code` (858)

54.19.66 unlock_data

Synopsis: Unlock data segment.

Declaration: `function unlock_data(var data; size: LongInt) : Boolean`

Visibility: default

Description: Unlocks a memory range which resides in the data segment selector.

Parameters:

data address of memory to be unlocked.

size size bytes to be unlocked.

Return values: `True` if successful, `False` otherwise.

For an example, see `get_rm_callback` (847).

Errors: Check the `int31error` (834) variable.

See also: `unlock_linear_region` (870), `unlock_code` (869), `lock_linear_region` (858), `lock_data` (858), `lock_code` (858)

54.19.67 unlock_linear_region

Synopsis: Unlock linear memory region.

Declaration: `function unlock_linear_region(linearaddr: LongInt; size: LongInt)
: Boolean`

Visibility: default

Description: Unlocks a previously locked linear region range to allow it to be swapped out again if needed.

Parameters:

linearaddr linear address of the memory to be unlocked.

size size bytes to be unlocked.

Return values: `True` if successful, `False` otherwise.

Errors: Check the `int31error` (834) variable.

See also: `unlock_data` (869), `unlock_code` (869), `lock_linear_region` (858), `lock_data` (858), `lock_code` (858)

54.20 tdpmiversioninfo

```
tdpmiversioninfo = record  
  major : Byte;  
  minor : Byte;  
  flags  
  : Word;  
  cpu : Byte;  
  master_pic : Byte;  
  slave_pic : Byte;  
end
```

`tdpmiversioninfo` describes the dpmi version information, as returned by `get_dpmi_version` (843). The CPU field can have the following values:

\$02H 80286

\$03H 80386

\$04H 80486

\$05H- Newer than 80486

The flags field is a bitmask with the following bits:

0 0 for 16 bit DPML, 1 for 32-bit

1 0 for virtual 86 mode for reflected interrupts, 1 for return to real mode.

2 0 for no virtual memory support, 1 for virtual memory support.

54.21 tmeminfo

```
tmeminfo = record
  available_memory : LongInt;
  available_pages
    : LongInt;
  available_lockable_pages : LongInt;
  linear_space
    : LongInt;
  unlocked_pages : LongInt;
  available_physical_pages
    : LongInt;
  total_physical_pages : LongInt;
  free_linear_space
    : LongInt;
  max_pages_in_paging_file : LongInt;
  reserved0 : LongInt
  ;
  reserved1 : LongInt;
  reserved2 : LongInt;
end
```

tmeminfo Holds information about the memory allocation, etc.

NOTE: The value of a field is -1 (0ffffffh) if the value is unknown, it's only guaranteed, that available_memory contains a valid value. The size of the pages can be determined by the get_page_size() function.

54.22 tseginfo

```
tseginfo = record
  offset : pointer;
  segment : Word;
end
```

This record is used to store a full 48-bit pointer. This may be either a protected mode selector:offset address or in real mode a segment:offset address, depending on application.

See also: Selectors and descriptors, dos memory access, Interrupt redirection

Chapter 55

Reference for unit 'gpm'

55.1 Used units

Table 55.1: Used units by unit 'gpm'

Name	Page
BaseUnix	155
System	1335

55.2 Overview

The GPM unit implements an interface to `libgpm`, the console program for mouse handling. This unit was created by Peter Vreman, and is only available on Linux.

When this unit is used, your program is linked to the C libraries, so you must take care of the C library version. Also, it will only work with version 1.17 or higher of the `libgpm` library.

55.3 Constants, types and variables

55.3.1 Constants

`GPM_BOT` = 2

Bottom of area.

`GPM_B_LEFT` = 4

Left mouse button identifier.

`GPM_B_MIDDLE` = 2

Middle mouse button identifier.

`GPM_B_RIGHT` = 1

Right mouse button identifier.

GPM_DOUBLE = 32

Mouse double click event.

GPM_DOWN = 4

Mouse button down event.

GPM_DRAG = 2

Mouse drag event.

GPM_ENTER = 512

Enter area event.

GPM_HARD = 256

?

GPM_LEAVE = 1024

Leave area event.

GPM_LEFT = 4

Left side of area.

GPM_MAGIC = \$47706D4C

Constant identifying GPM in Gpm_Open ([879](#)).

GPM_MFLAG = 128

Motion flag.

GPM_MOVE = 1

Mouse move event.

GPM_NODE_CTL = GPM_NODE_DEV

Control socket.

GPM_NODE_DEV = '/dev/gpmctl'

Device socket filename.

GPM_NODE_DIR = _PATH_VARRUN

Where to write socket.

```
GPM_NODE_DIR_MODE = 0775
```

Mode of socket.

```
GPM_NODE_FIFO = '/dev/gpmdata'
```

FIFO name.

```
GPM_NODE_PID = '/var/run/gpm.pid'
```

Name of PID file.

```
GPM_RGT = 8
```

Right side of area.

```
GPM_SINGLE = 16
```

Mouse single click event.

```
GPM_TOP = 1
```

Top of area.

```
GPM_TRIPLE = 64
```

Mouse triple click event.

```
GPM_UP = 8
```

Mouse button up event.

```
_PATH_DEV = '/dev/'
```

Location of `/dev` directory.

```
_PATH_VARRUN = '/var/run/'
```

Location of run PID files directory.

55.3.2 Types

```
Pgpmconnect = Pgpm_connect
```

Pointer to `TGpmConnect` (875) record.

```
Pgpmevent = Pgpm_event
```

Pointer to TGpmEvent (875) record.

```
Pgpmroi = Pgpm_roi
```

Pointer to TGpmRoi (875) record.

```
Pgpm_connect = ^TGpm_connect
```

Pointer to TGpm_Connect (882) record.

```
Pgpm_event = ^Tgpm_event
```

Pointer to TGpm_Event (882) record.

```
Pgpm_roi = ^Tgpm_roi
```

Pointer to Tgpm_roi (883) record.

```
Tgpmconnect = Tgpm_connect
```

Alias for TGpm_Connect (882) record.

```
TGpmEtype = LongInt
```

Type for event type.

```
Tgpmevent = Tgpm_event
```

Alias for TGPM_EVent (882) record.

```
TGpmHandler = function(var event: Tgpmevent; clientdata: pointer)
    : LongInt
```

Mouse event handler callback.

```
TGpmMargin = LongInt
```

Type to hold area margin.

```
Tgpmroi = Tgpm_roi
```

Alias for TGpm_roi (883)Record.

55.3.3 Variables

```
gpm_current_roi : Pgpm_roi
```

Internal gpm library variable. Do not use.

```
gpm_handler : TGpmHandler
```

Internal gpm library variable. Do not use.

`gpm_roi : Pgpm_roi`

Internal gpm library variable. Do not use.

`gpm_roi_data : pointer`

Internal gpm library variable. Do not use.

`gpm_roi_handler : TGpmHandler`

Internal gpm library variable. Do not use.

55.4 Procedures and functions

55.4.1 Gpm_AnyDouble

Synopsis: Check whether event has double click event.

Declaration: `function Gpm_AnyDouble(EventType: LongInt) : Boolean`

Visibility: default

Description: `Gpm_AnyDouble` returns True if `EventType` contains the `GPM_DOUBLE` flag, False otherwise.

Errors: None.

See also: `Gpm_StrictSingle` (881), `Gpm_AnySingle` (876), `Gpm_StrictDouble` (881), `Gpm_StrictTriple` (881), `Gpm_AnyTriple` (876)

55.4.2 Gpm_AnySingle

Synopsis: Check whether event has a single click event.

Declaration: `function Gpm_AnySingle(EventType: LongInt) : Boolean`

Visibility: default

Description: `Gpm_AnySingle` returns True if `EventType` contains the `GPM_SINGLE` flag, False otherwise.

Errors: None.

See also: `Gpm_StrictSingle` (881), `Gpm_AnyDouble` (876), `Gpm_StrictDouble` (881), `Gpm_StrictTriple` (881), `Gpm_AnyTriple` (876)

55.4.3 Gpm_AnyTriple

Synopsis: Check whether event has a triple click event.

Declaration: `function Gpm_AnyTriple(EventType: LongInt) : Boolean`

Visibility: default

Description: `Gpm_AnySingle` returns `True` if `EventType` contains the `GPM_TRIPLE` flag, `False` otherwise.

Errors: None.

See also: `Gpm_StrictSingle` (881), `Gpm_AnyDouble` (876), `Gpm_StrictDouble` (881), `Gpm_StrictTriple` (881), `Gpm_AnySingle` (876)

55.4.4 `gpm_close`

Synopsis: Close connection to GPM server.

Declaration: `function gpm_close : LongInt`

Visibility: `default`

Description: `Gpm_Close` closes the current connection, and pops the connection stack; this means that the previous connection becomes active again.

The function returns `-1` if the current connection is not the last one, and it returns `0` if the current connection is the last one.

for an example, see `Gpm_GetEvent` (878).

Errors: None.

See also: `Gpm_Open` (879)

55.4.5 `gpm_fitvalues`

Synopsis: Change coordinates to fit physical screen.

Declaration: `function gpm_fitvalues(var x: LongInt; var y: LongInt) : LongInt`

Visibility: `default`

Description: `Gpm_fitValues` changes `x` and `y` so they fit in the visible screen. The actual mouse pointer is not affected by this function.

Errors: None.

See also: `Gpm_FitValuesM` (877)

55.4.6 `gpm_fitvaluesM`

Synopsis: Change coordinates to fit margin.

Declaration: `function gpm_fitvaluesM(var x: LongInt; var y: LongInt; margin: LongInt) : LongInt`

Visibility: `default`

Description: `Gpm_FitValuesM` changes `x` and `y` so they fit in the margin indicated by `margin`. If `margin` is `-1`, then the values are fitted to the screen. The actual mouse pointer is not affected by this function.

Errors: None.

See also: `Gpm_FitValues` (877)

55.4.7 gpm_getevent

Synopsis: Get event from event queue.

Declaration: `function gpm_getevent (var event : Tgpm_event) : LongInt`

Visibility: default

Description: `Gpm_GetEvent` Reads an event from the file descriptor `gpm_fd`. This file is only for internal use and should never be called by a client application.

It returns 1 on success, and -1 on failure.

Errors: On error, -1 is returned.

See also: `Gpm_GetSnapshot` ([879](#))

Listing: `./examples/gpmex/gpmex.pp`

```

program gpmex;

{
  Example program to demonstrate the use of the gpm unit.
}

uses gpm;

var
  connect : TGPMConnect;
  event : tgmevent;

begin
  connect.EventMask:=GPM_MOVE or GPM_DRAG or GPM_DOWN or GPM_UP;
  connect.DefaultMask:=0;
  connect.MinMod:=0;
  connect.MaxMod:=0;
  if Gpm_Open(connect,0)=-1 then
    begin
      Writeln('No mouse handler present. ');
      Halt(1);
    end;
  Writeln('Click right button to end. ');
  Repeat
    gpm_getevent(Event);
    With Event do
      begin
        Write('Pos = (,X,',',Y,') Buttons : ( ');
        if (buttons and Gpm_b_left)<>0 then
          write('left ');
        if (buttons and Gpm_b_right)<>0 then
          write('right ');
        if (buttons and Gpm_b_middle)<>0 then
          Write('middle ');
        Write(') Event : ');
        Case EventType and $F of
          GPM_MOVE: write('Move');
          GPM_DRAG: write('Drag');
          GPM_DOWN: write('Down');
          GPM_UP: write('Up');
        end;
      end;
    until Event = GPM_UP;
  end;

```

```

        Writeln;
    end;
    Until (Event.Buttons and gpm_b_right) <> 0;
    gpm_close;
end.

```

55.4.8 gpm_getsnapshot

Synopsis: Return servers' current image of mouse state.

Declaration: `function gpm_getsnapshot(eptr: Pgpmevent) : LongInt`
`function gpm_getsnapshot(var eptr: Tgpmevent) : LongInt`

Visibility: default

Description: `Gpm_GetSnapshot` returns the picture that the server has of the current situation in `Event`. This call will not read the current situation from the mouse file descriptor, but returns a buffered version.

The function returns the number of mouse buttons, or -1 if this information is not available.

Errors: None.

See also: `Gpm_GetEvent` ([878](#))

55.4.9 gpm_lowerroi

Synopsis: Lower a region of interest in the stack.

Declaration: `function gpm_lowerroi(which: Pgpm_roi; after: Pgpm_roi) : Pgpm_roi`

Visibility: default

Description: `Gpm_LowerRoi` lowers the region of interest which after `after`. If `after` is `Nil`, the region of interest is moved to the bottom of the stack.

The return value is the new top of the region-of-interest stack.

Errors: None.

See also: `Gpm_RaiseRoi` ([880](#)), `Gpm_PopRoi` ([880](#)), `Gpm_PushRoi` ([880](#))

55.4.10 gpm_open

Synopsis: Open connection to GPM server.

Declaration: `function gpm_open(var conn: Tgpm_connect; flag: LongInt) : LongInt`

Visibility: default

Description: `Gpm_Open` opens a new connection to the mouse server. The connection is described by the fields of the `conn` record of type `TGPMConnect` ([875](#)).

if `Flag` is 0, then the application only receives events that come from its own terminal device. If it is negative it will receive all events. If the value is positive then it is considered a console number to which to connect.

The return value is -1 on error, or the file descriptor used to communicate with the client. Under an X-Term the return value is -2.

for an example, see `Gpm_GetEvent` ([878](#)).

Errors: On Error, the return value is -1.

See also: [Gpm_Open \(879\)](#)

55.4.11 gpm_poproi

Synopsis: Pop region of interest from the stack.

Declaration: `function gpm_poproi(which: Pgpm_roi) : Pgpm_roi`

Visibility: default

Description: `Gpm_PopRoi` pops the topmost region of interest from the stack. It returns the next element on the stack, or `Nil` if the current element was the last one.

Errors: None.

See also: [Gpm_RaiseRoi \(880\)](#), [Gpm_LowerRoi \(879\)](#), [Gpm_PushRoi \(880\)](#)

55.4.12 gpm_pushroi

Synopsis: Push region of interest on the stack.

Declaration: `function gpm_pushroi(x1: LongInt; y1: LongInt; x2: LongInt;
y2: LongInt; mask: LongInt; fun: TGpmHandler;
xtradata: pointer) : Pgpm_roi`

Visibility: default

Description: `Gpm_PushRoi` puts a new *region of interest* on the stack. The region of interest is defined by a rectangle described by the corners `(X1, Y1)` and `(X2, Y2)`.

The mask describes which events the handler {fun} will handle; `ExtraData` will be put in the `xtradata` field of the {TGPM_Roi} record passed to the fun handler.

Errors: None.

See also: [Gpm_RaiseRoi \(880\)](#), [Gpm_PopRoi \(880\)](#), [Gpm_LowerRoi \(879\)](#)

55.4.13 gpm_raiseroi

Synopsis: Raise region of interest in the stack.

Declaration: `function gpm_raiseroi(which: Pgpm_roi; before: Pgpm_roi) : Pgpm_roi`

Visibility: default

Description: `Gpm_RaiseRoi` raises the *region of interest* which till it is on top of region before. If before is nil then the region is put on top of the stack. The returned value is the top of the stack.

Errors: None.

See also: [Gpm_PushRoi \(880\)](#), [Gpm_PopRoi \(880\)](#), [Gpm_LowerRoi \(879\)](#)

55.4.14 **gpm_repeat**

Synopsis: Check for presence of mouse event.

Declaration: `function gpm_repeat (millisec: LongInt) : LongInt`

Visibility: default

Description: `Gpm_Repeat` returns 1 if no mouse event arrives in the next `millisec` milliseconds, it returns 0 otherwise.

Errors: None.

See also: `Gpm_GetEvent` ([878](#))

55.4.15 **Gpm_StrictDouble**

Synopsis: Check whether event contains only a double-click event.

Declaration: `function Gpm_StrictDouble (EventType: LongInt) : Boolean`

Visibility: default

Description: `Gpm_StrictDouble` returns true if `EventType` contains only a doubleclick event, False otherwise.

Errors: None.

See also: `Gpm_StrictSingle` ([881](#)), `Gpm_AnyTriple` ([876](#)), `Gpm_AnyDouble` ([876](#)), `Gpm_StrictTriple` ([881](#)), `Gpm_AnySingle` ([876](#))

55.4.16 **Gpm_StrictSingle**

Synopsis: Check whether event contains only a single-click event.

Declaration: `function Gpm_StrictSingle (EventType: LongInt) : Boolean`

Visibility: default

Description: `Gpm_StrictDouble` returns True if `EventType` contains only a singleclick event, False otherwise.

Errors: None.

See also: `Gpm_AnyTriple` ([876](#)), `Gpm_StrictDouble` ([881](#)), `Gpm_AnyDouble` ([876](#)), `Gpm_StrictTriple` ([881](#)), `Gpm_AnySingle` ([876](#))

55.4.17 **Gpm_StrictTriple**

Synopsis: Check whether event contains only a triple-click event.

Declaration: `function Gpm_StrictTriple (EventType: LongInt) : Boolean`

Visibility: default

Description: `Gpm_StrictTriple` returns true if `EventType` contains only a triple click event, False otherwise.

Errors: None.

See also: `Gpm_AnyTriple` ([876](#)), `Gpm_StrictDouble` ([881](#)), `Gpm_AnyDouble` ([876](#)), `Gpm_StrictSingle` ([881](#)), `Gpm_AnySingle` ([876](#))

55.5 Tgpm_connect

```
Tgpm_connect = record
  eventMask : Word;
  defaultMask : Word;
  minMod : Word;
  maxMod : Word;
  pid : LongInt;
  vc : LongInt;
end
```

GPM server connection information.

55.6 Tgpm_event

```
Tgpm_event = record
  buttons : Byte;
  modifiers : Byte;
  vc : Word
;
  dx : Word;
  dy : Word;
  x : Word;
  y : Word;
  EventType :
  TGpmEtype;
  clicks : LongInt;
  margin : TGpmMargin;
  wdx : Word
;
  wdy : Word;
end
```

Tgpm_event describes the events that are reported by GPM.

55.7 Tgpm_roi

```
Tgpm_roi = record
  xmin : Integer;
  xmax : Integer;
  ymin : Integer
;
  ymax : Integer;
  minmod : Word;
  maxmod : Word;
  eventmask
  : Word;
  owned : Word;
  handler : TGpmHandler;
  clientdata : pointer
;
```

```
prev : Pgpm_roi;  
next : Pgpm_roi;  
end
```

Record used to define regions of interest.

Chapter 56

Reference for unit 'Graph'

56.1 Used units

Table 56.1: Used units by unit 'Graph'

Name	Page
System	1335

56.2 Overview

This document describes the `GRAPH` unit for Free Pascal, for all platforms. The unit was first written for dos by Florian Klaempfl, but was later completely rewritten by Carl-Eric Codere to be completely portable. The unit is provided for compatibility only: It is recommended to use more modern graphical systems. The graph unit will allow to recompile old programs. They will work to some extent, but if the application has heavy graphical needs, it's recommended to use another set of graphical routines, suited to the platform the program should work on.

56.3 Categorized functions: Text and font handling.

Functions to set texts on the screen.

Table 56.2:

Name	Description
GetTextSettings (923)	Get current text settings
InstallUserFont (926)	Install a new font
OutText (927)	Write text at current cursor position
OutTextXY (915)	Write text at coordinates X,Y
RegisterBGIFont (929)	Register a new font
SetTextJustify (932)	Set text justification
SetTextStyle (932)	Set text style
SetUserCharSize (933)	Set text size
TextHeight (934)	Calculate height of text
TextWidth (934)	Calculate width of text

56.4 Categorized functions: Filled drawings.

Functions for drawing filled regions.

Table 56.3:

Name	Description
Bar3D (916)	Draw a filled 3D-style bar
Bar (916)	Draw a filled rectangle
FloodFill (918)	Fill starting from coordinate
FillEllipse (918)	Draw a filled ellipse
FillPoly (918)	Draw a filled polygon
GetFillPattern (920)	Get current fill pattern
GetFillSettings (920)	Get current fill settings
SetFillPattern (930)	Set current fill pattern
SetFillStyle (930)	Set current fill settings

56.5 Categorized functions: Drawing primitives.

Functions for simple drawing.

Table 56.4:

Name	Description
Arc (916)	Draw an arc
Circle (913)	Draw a complete circle
DrawPoly (917)	Draw a polygon with N points
Ellipse (918)	Draw an ellipse
GetArcCoords (919)	Get arc coordinates
GetLineSettings (921)	Get current line drawing settings
Line (915)	Draw line between 2 points
LineRel (926)	Draw line relative to current position
LineTo (927)	Draw line from current position to absolute position
MoveRel (927)	Move cursor relative to current position
MoveTo (927)	Move cursor to absolute position
PieSlice (928)	Draw a pie slice
PutPixel (915)	Draw 1 pixel
Rectangle (928)	Draw a non-filled rectangle
Sector (929)	Draw a sector
SetLineStyle (931)	Set current line drawing style

56.6 Categorized functions: Color management.

All functions related to color management.

Table 56.5:

Name	Description
GetBkColor (914)	Get current background color
GetColor (919)	Get current foreground color
GetDefaultPalette (919)	Get default palette entries
GetMaxColor (921)	Get maximum valid color
GetPaletteSize (923)	Get size of palette for current mode
GetPixel (914)	Get color of selected pixel
GetPalette (923)	Get palette entry
SetAllPalette (915)	Set all colors in palette
SetBkColor (915)	Set background color
SetColor (930)	Set foreground color
SetPalette (932)	Set palette entry
SetRGBPalette (916)	Set palette entry with RGB values

56.7 Categorized functions: Screen management.

General drawing screen management functions.

Table 56.6:

Name	Description
ClearViewPort (914)	Clear the current viewport
GetImage (914)	Copy image from screen to memory
GetMaxX (922)	Get maximum X coordinate
GetMaxY (922)	Get maximum Y coordinate
GetX (924)	Get current X position
GetY (924)	Get current Y position
ImageSize (914)	Get size of selected image
GetViewSettings (923)	Get current viewport settings
PutImage (915)	Copy image from memory to screen
SetActivePage (915)	Set active video page
SetAspectRatio (929)	Set aspect ratio for drawing routines
SetViewPort (933)	Set current viewport
SetVisualPage (916)	Set visual page
SetWriteMode (934)	Set write mode for screen operations

56.8 Categorized functions: Initialization.

Initialization of the graphics screen.

Table 56.7:

Name	Description
ClearDevice (917)	Empty the graphics screen
CloseGraph (917)	Finish drawing session, return to text mode
DetectGraph (917)	Detect graphical modes
GetAspectRatio (919)	Get aspect ratio of screen
GetModeRange (922)	Get range of valid modes for current driver
GraphDefaults (924)	Set defaults
GetDriverName (920)	Return name of graphical driver
GetGraphMode (921)	Return current or last used graphics mode
GetMaxMode (921)	Get maximum mode for current driver
GetModeName (922)	Get name of current mode
GraphErrorMsg (924)	String representation of graphical error
GraphResult (925)	Result of last drawing operation
InitGraph (925)	Initialize graphics drivers
InstallUserDriver (926)	Install a new driver
RegisterBGIDriver (928)	Register a new driver
RestoreCRTMode (929)	Go back to text mode
SetGraphMode (931)	Set graphical mode

56.9 Target specific issues: Linux.

There are several issues on Linux that need to be taken care of:

The Linux version of the Graph unit uses the libvga library. This library works on the console, not under X.

If you get an error similar to

```
/usr/bin/ld: cannot find -lvga
```

This can mean one of two things: either `libvga` and its development package is not installed properly, or the directory where it is installed is not in the linker path.

To remedy the former, you should install both the `libvga` package and `libvga-devel` package (or compile and install from scratch).

To remedy the latter, you should add the path to the compiler command-line using the `-F1` option.

Programs using `libvga` need root privileges to run. You can make them `setuid` root with the following command:

```
chown root.root myprogram
chmod u+s myprogram
```

The `libvga` library will give up the root privileges after it is initialized.

there is an experimental version of the Graphics library available that uses GGI to do all the drawing, but it is not well tested. It's called `ggigraph` and is distributed in source form only.

Do not use the CRT unit together with the Graph unit: the console may end up in an unusable state. Instead, the `ncurses` unit may function fine.

56.10 Target specific issues: DOS.

VESA modes (i.e., anything but 320x200x256 and 640x480x16) do not work under most installations of Windows NT, Windows 2000 and Windows XP. They also do not work for some people under Windows 98 and Windows ME, depending on their graphics drivers. However, the graph unit cannot detect this, because no errors are returned from the system. In such cases, the screen simply turns black, or will show garbage.

Nothing can be done about this, the reason is missing or buggy support in the graphics drivers of the operating system.

56.11 A word about mode selection.

The graph unit was implemented for compatibility with the old Turbo Pascal graph unit. For this reason, the mode constants as they were defined in the Turbo Pascal graph unit are retained.

However, since

1. Video cards have evolved very much
2. Free Pascal runs on multiple platforms

it was decided to implement new mode and graphic driver constants, which are more independent of the specific platform the program runs on.

In this section we give a short explanation of the new mode system. the following drivers were defined:

```
D1bit = 11;
D2bit = 12;
```

```

D4bit = 13;
D6bit = 14; { 64 colors Half-brite mode - Amiga }
D8bit = 15;
D12bit = 16; { 4096 color modes HAM mode - Amiga }
D15bit = 17;
D16bit = 18;
D24bit = 19; { not yet supported }
D32bit = 20; { not yet supported }
D64bit = 21; { not yet supported }

lowNewDriver = 11;
highNewDriver = 21;

```

Each of these drivers specifies a desired color-depth.

The following modes have been defined:

```

detectMode = 30000;
m320x200 = 30001;
m320x256 = 30002; { amiga resolution (PAL) }
m320x400 = 30003; { amiga/atari resolution }
m512x384 = 30004; { mac resolution }
m640x200 = 30005; { vga resolution }
m640x256 = 30006; { amiga resolution (PAL) }
m640x350 = 30007; { vga resolution }
m640x400 = 30008;
m640x480 = 30009;
m800x600 = 30010;
m832x624 = 30011; { mac resolution }
m1024x768 = 30012;
m1280x1024 = 30013;
m1600x1200 = 30014;
m2048x1536 = 30015;

lowNewMode = 30001;
highNewMode = 30015;

```

These modes start at 30000 because Borland specified that the mode number should be ascending with increasing X resolution, and the new constants shouldn't interfere with the old ones.

The above constants can be used to set a certain color depth and resolution, as demonstrated in the below example.

If other modes than the ones above are supported by the graphics card, you will not be able to select them with this mechanism.

For this reason, there is also a 'dynamic' mode number, which is assigned at run-time. This number increases with increasing X resolution. It can be queried with the `getmoderange` call. This call will return the range of modes which are valid for a certain graphics driver. The numbers are guaranteed to be consecutive, and can be used to search for a certain resolution, as in the second example below.

Thus, the `getmoderange` function can be used to detect all available modes and drivers, as in the third example below:

Listing: ./examples/graphex/inigraph1.pp

Program inigraph1;

```
{ Program to demonstrate static graphics mode selection }
```

```
uses graph;
```

```
const
```

```
  TheLine = 'We are now in 640 x 480 x 256 colors!'+  
            ' (press <Return> to continue)';
```

```
var
```

```
  gd, gm, lo, hi, error, tw, th: integer;  
  found: boolean;
```

```
begin
```

```
  { We want an 8 bit mode }  
  gd := D8bit;  
  gm := m640x480;  
  initgraph(gd, gm, "");  
  { Make sure you always check graphresult! }  
  error := graphResult;  
  if (error <> grOk) Then  
    begin  
    writeln('640x480x256 is not supported!');  
    halt(1)  
    end;  
  { We are now in 640x480x256 }  
  setColor(cyan);  
  rectangle(0,0,getmaxx,getmaxy);  
  { Write a nice message in the center of the screen }  
  setTextStyle(defaultFont, horizDir, 1);  
  tw := TextWidth(TheLine);  
  th := TextHeight(TheLine);  
  outTextXY((getMaxX - TW) div 2,  
            (getMaxY - TH) div 2, TheLine);  
  { Wait for return }  
  readln;  
  { Back to text mode }  
  closegraph;  
end.
```

Listing: ./examples/graphex/inigraph2.pp

Program inigraph2;

```
{ Program to demonstrate dynamic graphics mode selection }
```

```
uses graph;
```

```
const
```

```
  TheLine = 'We are now in 640 x 480 x 256 colors!'+  
            ' (press <Return> to continue)';
```

```
var
```

```
  th, tw, gd, gm, lo, hi, error: integer;  
  found: boolean;
```

```
begin
```

```
  { We want an 8 bit mode }
```

```

gd := D8bit;
{ Get all available resolutions for this bitdepth }
getmoderange(gd,lo,hi);
{ If the highest available mode number is -1,
  no resolutions are supported for this bitdepth }
if hi = -1 then
  begin
    writeln('no 8 bit modes supported!');
    halt
  end;
found := false;
{ Search all resolutions for 640x480 }
for gm := lo to hi do
  begin
    initgraph(gd,gm,"");
    { Make sure you always check graphresult! }
    error := graphResult;
    if (error = grOk) and
      (getmaxx = 639) and (getmaxy = 479) then
      begin
        found := true;
        break;
      end;
  end;
if not found then
  CloseGraph();
  begin
    writeln('640x480x256 is not supported!');
    halt(1)
  end;
{ We are now in 640x480x256 }
setColor(cyan);
rectangle(0,0,getmaxx,getmaxy);
{ Write a nice message in the center of the screen }
setTextStyle(defaultFont,horizDir,1);
TW:=TextWidth(TheLine);
TH:=TextHeight(TheLine);
outTextXY((getMaxX - TW) div 2,
          (getMaxY - TH) div 2,TheLine);
{ Wait for return }
readln;
{ Back to text mode }
closegraph;
end.

```

Listing: ./examples/graphex/modrange.pp

Program GetModeRange_Example;

{ This program demonstrates how to find all available graph modes }

uses graph;

const

*{ Currently, only 4, 8, 15 and 16 bit modes are supported
but this may change in the future }*
 gdnames: **array**[D4bit..D16bit] **of string**[6] =
 ('4 bit','6 bit','8 bit','12 bit','15 bit','16 bit');

```

procedure WriteRes(const depth : integer);
var
    tw, th : integer;
    v, text : String;
begin
    text := 'Current resolution is '; str(getmaxx+1, v);
    text := text + v + 'x'; str(getmaxy+1, v);
    text := text + v + 'x' + gdnames[depth];
    setTextStyle(defaultFont, horizDir, 1);
    TW:=TextWidth(text);
    TH:=TextHeight(text);
    outTextXY((getMaxX - TW) div 2,
              (getMaxY - TH) div 2, text);
end;

var
    t: text;
    line : string;
    gd, c, low, high, res: integer;
begin
    assign(t, 'modes.txt');
    rewrite(t);
    close(t);
    for gd := D4bit to D16bit do
        begin
            { Get the available mode numbers for this driver }
            getModeRange(gd, low, high);
            append(t);
            write(t, gdnames[gd]);
            Writeln(t, ': low modenr = ', low, ', high modenr = ', high);
            close(t);
            { If high is -1,
              no resolutions are supported for this bitdepth }
            if high = -1 then
                begin
                    append(t);
                    writeln(t, ' No modes supported!');
                    writeln(t);
                    close(t);
                end
            else
                { Enter all supported resolutions for this bitdepth
                  and write their characteristics to the file }
                for c := low to high do
                    begin
                        append(t);
                        writeln(t, ' testing mode nr ', c);
                        close(t);
                        initgraph(gd, c, '');
                        res := graphresult;
                        append(t);
                        { An error occurred when entering the mode? }
                        if res <> grok then
                            writeln(t, grapherrormsg(res))
                        else
                            begin
                                write(t, 'maxx: ', getmaxx, ', maxy: ', getmaxy);

```

```

        Writeln(t, ' maxcolor: ', getmaxcolor);
        closegraph;
    end;
    writeln(t);
    WriteRes(gd);
    close(t);
    end;
    append(t);
    writeln(t);
    close(t);
    end;
    Writeln('All supported modes are listed in modes.txt files');
end.

```

56.12 Requirements.

The unit Graph exports functions and procedures for graphical output. It requires at least a VGA-compatible Card or a VGA-Card with software-driver (min. **512Kb** video memory).

56.13 Constants, types and variables

56.13.1 Constants

AndPut = 3

Draw operation: use AND.

```

AnsiToASCIITransTable : TCharsetTransTable = (#$00, #$01, #$02, #
    $03, #$04, #$05, #$06, #$07, #$08, #$09, #$0a, #$0b, #$0c, #$0d,
    #$0e, #$0f, #$10, #$11, #$12, #$13, #$14, #$15, #$16, #$17, #$18,
    #$19, #$1a, #$1b, #$1c, #$1d, #$1e, #$1f, #$20, #$21, #$22, #$23
    , #$24, #$25, #$26, #$27, #$28, #$29, #$2a, #$2b, #$2c, #$2d, #$2e
    , #$2f, #$30, #$31, #$32, #$33, #$34, #$35, #$36, #$37, #$38, #$39
    , #$3a, #$3b, #$3c, #$3d, #$3e, #$3f, #$40, #$41, #$42, #$43, #$44
    , #$45, #$46, #$47, #$48, #$49, #$4a, #$4b, #$4c, #$4d, #$4e, #$4f
    , #$50, #$51, #$52, #$53, #$54, #$55, #$56, #$57, #$58, #$59, #$5a
    , #$5b, #$5c, #$5d, #$5e, #$5f, #$60, #$61, #$62, #$63, #$64, #$65
    , #$66, #$67, #$68, #$69, #$6a, #$6b, #$6c, #$6d, #$6e, #$6f, #$70
    , #$71, #$72, #$73, #$74, #$75, #$76, #$77, #$78, #$79, #$7a, #$7b
    , #$7c, #$7d, #$7e, #$7f, '?', '?', '?', '?', '?', '?', '?', '?',
    '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?',
    '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '$ff', $ad
    , $9b, $9c, '?', $9d, '?', '?', '?', '?', $a6, $ae, $aa, '?'
    , '?', '?', $f8, $f1, $fd, '?', '?', $e6, '?', $fa, '?', '?'
    , $a7, $af, $ac, $ab, '?', $a8, '?', '?', '?', '?', $8e, $8f
    , $92, $80, '?', $90, '?', '?', '?', '?', '?', '?', '?', $a5,
    '?', '?', '?', '?', $99, '?', '?', '?', '?', '?', $9a, '?', '?'
    , $e1, $85, $a0, $83, '?', $84, $86, $91, $87, $8a, $82
    , $88, $89, $8d, $a1, $8c, $8b, '?', $a4, $95, $a2, $93
    , '?', $94, $f6, '?', $97, $a3, $96, $81, '?', '?', $98)

```

Default ansi transliteration table.

`BkSlashFill = 5`

Fill style: Diagonal (backslash) lines.

`black = 0`

Color code: black.

`blue = 1`

Color code: blue.

`BoldFont = 10`

Font number: Bold font.

`BottomText = 0`

Vertical text alignment: Align text to bottom.

`brown = 6`

Color code: brown.

`CenterLn = 2`

Line style: centered line.

`CenterText = 1`

Horizontal text alignment: Center text.

`CGA = 1`

Graphic driver for CGA cards.

`CGAC0 = 0`

CGA Graphic driver mode C0.

`CGAC1 = 1`

CGA Graphic driver mode C1.

`CGAC2 = 2`

CGA Graphic driver mode C2.

`CGAC3 = 3`

CGA Graphic driver mode C3.

CGAHi = 4

CGA Graphic driver Hi-res mode.

ClipOff = False

Viewport clipping off.

ClipOn = True

Viewport clipping on.

CloseDotFill = 11

Fill style: Closely spaced dotted lines.

CopyPut = 0

Draw operation: use Copy.

CurrentDriver = - 128

Currently used driver.

cyan = 3

Color code: Cyan.

D12bit = 16

Mode: Depth 12 bit.

D15bit = 17

Mode: Depth 15 bit.

D16bit = 18

Mode: Depth 16 bit.

D1bit = 11

Mode: Depth 1 bit.

D24bit = 19

Mode: Depth 24 bit.

D2bit = 12

Mode: Depth 2 bit.

D32bit = 20

Mode: Depth 32 bit.

D4bit = 13

Mode: Depth 4 bit.

D64bit = 21

Mode: Depth 64 bit.

D6bit = 14

Mode: Depth 6 bit.

D8bit = 15

Mode: Depth 8 bit.

darkgray = 8

Color code: Dark gray.

DashedLn = 3

Line style: dashed line.

Default = 0

Default mode.

DefaultFont = 0

Font number: Normal font.

Detect = 0

Mode: Detect mode.

detectMode = 30000

Mode: Autodetect optimal mode.

DottedLn = 1

Line style: Dotted line.

DrawTextBackground : Boolean = False

Should the background of texts be drawn or should it be left untouched ?

EGA = 3

Graphic driver for EGA cards.

EGA64 = 4

Graphic driver for EGA 64 cards.

EGA64Hi = 1

EGA64 graphic driver high resolution mode.

EGA64Lo = 0

EGA64 graphic driver low resolution mode.

EGABlack = 0

Color code: EGA Black.

EGABlue = 1

Color code: EGA blue.

EGABrown = 20

Color code: EGA brown.

EGACyan = 3

Color code: EGA cyan.

EGADarkgray = 56

Color code: EGA dark gray.

EGAGreen = 2

Color code: EGA green.

EGAHi = 1

EGA graphic driver high resolution mode.

EGALightblue = 57

Color code: EGA Light blue.

EGALightcyan = 59

Color code: EGA Light cyan.

EGALightgray = 7

Color code: EGA Light gray.

EGALightgreen = 58

Color code: EGA Light green.

EGALightmagenta = 61

Color code: EGA light magenta.

EGALightred = 60

Color code: EGA light red.

EGALo = 0

EGA graphic driver low resolution mode.

EGAMagenta = 5

Color code: EGA magenta.

EGAMono = 5

Graphic driver for EGA monochrome cards.

EGAMonoHi = 3

EGAMono graphic driver high resolution mode.

EGARed = 4

Color code: EGA red.

EGAWhite = 63

Color code: EGA white.

EGAYellow = 62

Color code: EGA yellow.

EmptyFill = 0

Fill style: Do not fill.

EuroFont = 9

Font number: ?

```
fillpatternTable : Array[0..12] of FillPatternType = (($00, $00,
    $00, $00, $00, $00, $00, $00), ($ff, $ff, $ff, $ff, $ff, $ff, $ff
    , $ff), ($ff, $ff, $00, $00, $ff, $ff, $00, $00), ($01, $02, $04,
    $08, $10, $20, $40, $80), ($07, $0e, $1c, $38, $70, $e0, $c1, $83
    ), ($07, $83, $c1, $e0, $70, $38, $1c, $0e), ($5a, $2d, $96, $4b,
    $a5, $d2, $69, $b4), ($ff, $88, $88, $88, $ff, $88, $88, $88), (
    $18, $24, $42, $81, $81, $42, $24, $18), ($cc, $33, $cc, $33, $cc
    , $33, $cc, $33), ($80, $00, $08, $00, $80, $00, $08, $00), ($88,
    $00, $22, $00, $88, $00, $22, $00), (0, 0, 0, 0, 0, 0, 0, 0))
```

Table with standard fill patterns.

G1024x768x16 = 30

Mode: Resolution 1024x768, 16 colors.

G1024x768x16M = 25

Mode: Resolution 1024x768, 16M colors.

G1024x768x16M32 = 36

Mode: Resolution 1024x758, 16M 32-bit colors.

G1024x768x256 = 12

Mode: Resolution 1024x768, 256 colors.

G1024x768x32K = 23

Mode: Resolution 1024x768, 32K colors.

G1024x768x64K = 24

Mode: Resolution 1024x768, 64K colors.

G1152x864x16 = 38

Mode: Resolution 1152x864, 16 colors.

G1152x864x16M = 42

Mode: Resolution 1152x864, 16M colors.

G1152x864x16M32 = 43

Mode: Resolution 1152x864, 16M 32-bitcolors.

G1152x864x256 = 39

Mode: Resolution 1152x864, 256 colors.

G1152x864x32K = 40

Mode: Resolution 1152x864, 32K colors.

G1152x864x64K = 41

Mode: Resolution 1152x864, 64K colors.

G1280x1024x16 = 31

Mode: Resolution 1280x1024, 16 colors.

G1280x1024x16M = 28

Mode: Resolution 1280x1024, 16M colors.

G1280x1024x16M32 = 37

Mode: Resolution 1280x1024, 16M 32-bit colors.

G1280x1024x256 = 13

Mode: Resolution 1280x1024, 256 colors.

G1280x1024x32K = 26

Mode: Resolution 1280x1024, 32K colors.

G1280x1024x64K = 27

Mode: Resolution 1280x1024, 64K colors.

G1600x1200x16 = 44

Mode: Resolution 1600x1200, 16 colors.

G1600x1200x16M = 48

Mode: Resolution 1600x1200, 16M colors.

G1600x1200x16M32 = 49

Mode: Resolution 1600x1200, 16M 32-bit colors.

G1600x1200x256 = 45

Mode: Resolution 1600x1200, 256 colors.

G1600x1200x32K = 46

Mode: Resolution 1600x1200, 32K colors.

G1600x1200x64K = 47

Mode: Resolution 1600x1200, 64K colors.

G320x200x16 = 1

Mode: Resolution 320x200, 16 colors.

G320x200x16M = 16

Mode: Resolution 320x200, 16M colors.

G320x200x16M32 = 33

Mode: Resolution 320x200, 16M 32-bit colors.

G320x200x256 = 5

Mode: Resolution 320x200, 256 colors.

G320x200x32K = 14

Mode: Resolution 320x200, 32K colors.

G320x200x64K = 15

Mode: Resolution 320x200, 64K colors.

G320x240x256 = 6

Mode: Resolution 320x240, 256 colors.

G320x400x256 = 7

Mode: Resolution 320x400, 256 colors.

G360x480x256 = 8

Mode: Resolution 360x480, 256 colors.

G640x200x16 = 2

Mode: Resolution x, colors.

G640x350x16 = 3

Mode: Resolution x, colors.

G640x480x16 = 4

Mode: Resolution x, colors.

G640x480x16M = 19

Mode: Resolution 640x480, 16M colors.

G640x480x16M32 = 34

Mode: Resolution 640x480, 16M 32-bit colors.

G640x480x2 = 9

Mode: Resolution 640x480, 2 colors.

G640x480x256 = 10

Mode: Resolution 640x480, 256 colors.

G640x480x32K = 17

Mode: Resolution 640x480, 32K colors.

G640x480x64K = 18

Mode: Resolution 640x480, 64K colors.

G720x348x2 = 32

Mode: Resolution 720x348, 2 colors.

G800x600x16 = 29

Mode: Resolution 800x600, 16 colors.

G800x600x16M = 22

Mode: Resolution 800x600, 16M colors.

G800x600x16M32 = 35

Mode: Resolution 800x600, 16M 32-bit colors.

G800x600x256 = 11

Mode: Resolution 800x600, 256 colors.

G800x600x32K = 20

Mode: Resolution 800x600, 32K colors.

G800x600x64K = 21

Mode: Resolution 800x600, 64K colors.

`GothicFont = 4`

Font number: Gothic font.

`GraphStringTransTable : PCharsetTransTable = Nil`

Table used when transliterating strings.

`green = 2`

Color code: green.

`grError = - 11`

Error: Unknown error.

`grFileNotFound = - 3`

Error: File for driver not found.

`grFontNotFound = - 8`

Error: font description file not found.

`grInvalidDriver = - 4`

Error: Invalid driver specified.

`grInvalidFont = - 13`

Error: Invalid font description.

`grInvalidFontNum = - 14`

Error: Invalid font number.

`grInvalidMode = - 10`

Error: Invalid mode specified.

`grInvalidVersion = - 18`

Error: Invalid version.

`grIOerror = - 12`

Error: Unspecified Input/Output error.

`grNoFloodMem = - 7`

Error: Could not allocate memory for flood operation.

`grNoFontMem = - 9`

Error: Not enough memory to load font.

`grNoInitGraph = - 1`

Error: Graphical system not initialized.

`grNoLoadMem = - 5`

Error: Memory error.

`grNoScanMem = - 6`

Error: Could not allocate memory for scan.

`grNotDetected = - 2`

Error: Graphics device not detected.

`grOk = 0`

Graphical operation went OK.

`HatchFill = 7`

Fill style: Hatch lines.

`HercMono = 7`

Mode: Hercules, mono color.

`HercMonoHi = 0`

Mode: Hercules card, monochrome, high resolution.

`highNewDriver = 21`

Mode: highest number for new driver.

`highNewMode = m2048x1536`

Mode: Highest possible value of the new modes.

`HorizDir = 0`

Text write direction: Horizontal.

`InterleaveFill = 9`

Fill style: Interleaving lines.

LCOMFont = 8

Font number: ?

LeftText = 0

Horizontal text alignment: Align text left.

lightblue = 9

Color code: Light blue.

lightcyan = 11

Color code: Light cyan.

lightgray = 7

Color code: Light gray.

lightgreen = 10

Color code: Light green.

lightmagenta = 13

Color code: Light magenta.

lightred = 12

Color code: Light red.

LineFill = 2

Fill style: Fill using horizontal lines.

lowNewDriver = 11

Mode: lowest number for new driver.

lowNewMode = m320x200

Mode: Lowest possible value of the new modes.

LowRes = 6

Mode: Low resolution.

LtBkSlashFill = 6

Fill style: Light diagonal (backslash) lines.

LtSlashFill = 3

Fill style: Light diagonal (slash) lines.

m1024x768 = detectMode + 12

Mode: Resolution 1024x768.

m1280x1024 = detectMode + 13

Mode: Resolution 1280x1024.

m1600x1200 = detectMode + 14

Mode: Resolution 1600x1200.

m2048x1536 = detectMode + 15

Mode: Resolution 2048x1536.

m320x200 = detectMode + 1

Mode: Resolution 320x200.

m320x256 = detectMode + 2

Mode: Resolution 320x256.

m320x400 = detectMode + 3

Mode: Resolution 320x400.

m512x384 = detectMode + 4

Mode: Resolution 512x384.

m640x200 = detectMode + 5

Mode: Resolution 640x200.

m640x256 = detectMode + 6

Mode: Resolution 640x256.

m640x350 = detectMode + 7

Mode: Resolution 640x350.

m640x400 = detectMode + 8

Mode: Resolution 640x400.

`m640x480 = detectMode + 9`

Mode: Resolution 640x480.

`m800x600 = detectMode + 10`

Mode: Resolution 800x600.

`m832x624 = detectMode + 11`

Mode: Resolution 832x624.

`magenta = 5`

Color code: Magenta.

`MaxColors = 255`

Max amount of colors in a palette.

`maxsmallint = high(smallint)`

Maximum value for smallint type.

`MCGA = 2`

Graphic driver for MCGA cards.

`MCGAC0 = 0`

MCGA Graphic driver mode C0.

`MCGAC1 = 1`

MCGA Graphic driver mode C1.

`MCGAC2 = 2`

MCGA Graphic driver mode C2.

`MCGAC3 = 3`

MCGA Graphic driver mode C3.

`MCGAHi = 5`

MCGA Graphic driver high resolution mode.

`MCGAMed = 4`

MCGA Graphic driver medium resolution mode.

NormalPut = 0

Draw operation: Use Normal (copy) operation.

NormWidth = 1

Line width: Normal width.

NotPut = 4

Draw operation: use NOT.

OrPut = 2

Draw operation: use OR.

red = 4

Color code: Red.

```
resolutions : Array[lowNewMode..highNewMode] of TResolutionRec =
  ((x: 320; y: 200), (x: 320; y: 256), (x: 320; y: 400), (x: 512; y
   : 384), (x: 640; y: 200), (x: 640; y: 256), (x: 640; y: 350), (x:
   640; y: 400), (x: 640; y: 480), (x: 800; y: 600), (x: 832; y: 624
   ), (x: 1024; y: 768), (x: 1280; y: 1024), (x: 1600; y: 1200), (x:
   2048; y: 1536))
```

Array with actual resolutions of the new modes.

RightText = 2

Horizontal text alignment: Align text right.

SansSerifFont = 3

Font number: Sans Serif font.

ScriptFont = 5

Font number: Script font.

SimpleFont = 6

Font number: Simple font.

SlashFill = 4

Fill style: Diagonal (slash) lines.

SmallFont = 2

Font number: Small font.

`SolidFill = 1`

Fill style: Solid fill.

`SolidLn = 0`

Line style: Solid line.

`ThickWidth = 3`

Line width: double width.

`TopOff = False`

Top off.

`TopOn = True`

Top on.

`TopText = 2`

Vertical text alignment: Align text to top.

`TriplexFont = 1`

Font number: Triplex font.

`TSCRFont = 7`

Font number: Terminal font.

`UserBitLn = 4`

Line style: User defined.

`UserCharSize = 0`

User character size.

`UserFill = 12`

Fill style: User-defined fill.

`VertDir = 1`

Text write direction: Vertical.

`VESA = 10`

Mode: VESA graphics adaptor.

VGA = 9

Mode: VGA graphics adaptor.

VGAHi = 2

Mode: VGA high resolution (640x480).

VGALo = 0

Mode: VGA low resolution (640x200).

VGAMed = 1

Mode: VGA medium resolution (640x350).

white = 15

Color code: White.

WideDotFill = 10

Fill style: Widely spaced dotted lines.

XHatchFill = 8

Fill style: Heavy hatch lines.

XORPut = 1

Draw operation: use XOR.

yellow = 14

Color code: Yellow.

56.13.2 Types

CircleProc = procedure(X: SmallInt; Y: SmallInt; Radius: Word)

Standard circle drawing routine prototype.

clrviewproc = procedure

Standard clearviewport routine prototype.

ColorType = Word

Color type alias.

```
defpixelproc = procedure(X: SmallInt; Y: SmallInt)
```

This is the standard putpixel routine used by all function drawing routines, it will use the viewport settings, as well as clip, and use the current foreground color to plot the desired pixel.

```
ellipseproc = procedure(X: SmallInt; Y: SmallInt; XRadius: Word;
    YRadius: Word; stAngle: Word; EndAngle: Word
    ;
    fp: patternlineproc)
```

Standard ellipse drawing routine prototype.

```
FillPatternType = Array[1..8] of Byte
```

Bit pattern used when drawing lines. Set bits are drawn.

```
GetBkColorProc = function : ColorType
```

GetBkColorProc is the procedure prototype for the GetBkColor (914) method handler in TMod-
eInfo (937). The function should return the color code of the background color.

```
getimageproc = procedure(X1: SmallInt; Y1: SmallInt; X2: SmallInt
    ;
    Y2: SmallInt; var Bitmap)
```

Standard GetImage (914) procedure prototype.

```
getpixelproc = function(X: SmallInt; Y: SmallInt) : ColorType
```

Standard pixel fetching routine prototype.

```
getrgbpaletteproc = procedure(ColorNum: SmallInt;
    var RedValue: SmallInt;
    var GreenValue: SmallInt;
    var
    BlueValue: SmallInt)
```

This routine prototype is a hook for GetRGBPalette (914).

```
getscanlineproc = procedure(X1: SmallInt; X2: SmallInt; Y: SmallInt
    ;
    var data)
```

This routine is used for FloodFill (918) It returns an entire screen scan line with a word for each pixel in the scanline. Also handy for GetImage.

```
graphfreememprc = procedure(var P: Pointer; size: Word)
```

Procedure prototype, used when heap memory is freed by the graph routines.


```
graphgetmemproc = procedure(var P: pointer; size: Word)
```

Procedure prototype, used when heap memory is needed by the graph routines.

```
graph_float = single
```

The platform's preferred floating point size for fast graph operations.

```
hlineproc = procedure(x: SmallInt; x2: SmallInt; y: SmallInt)
```

Standard procedure prototype to draw a single horizontal line.

```
imagesizeproc = function(X1: SmallInt; Y1: SmallInt; X2: SmallInt
;
                        Y2: SmallInt) : LongInt
```

Standard ImageSize (914) calculation procedure prototype.

```
initmodeproc = procedure
```

Standard routine prototype to initialize a mode.

```
lineproc = procedure(X1: SmallInt; Y1: SmallInt; X2: SmallInt;
Y2: SmallInt)
```

Standard line drawing routine prototype.

```
OutTextXYProc = procedure(x: SmallInt; y: SmallInt;
const TextString: string)
```

This routine prototype is a hook for OutTextXY (915).

```
patternlineproc = procedure(x1: SmallInt; x2: SmallInt; y: SmallInt
)
```

Standard procedure prototype to draw a patterned line.

```
PCharsetTransTable = ^TCharsetTransTable
```

Pointer to TCharsetTransTable (913) array.

```
PModeInfo = ^TModeInfo
```

Pointer to TModeInfo (937) record.

```
putimageproc = procedure(X: SmallInt; Y: SmallInt; var Bitmap;
BitBlt: Word)
```

Standard PutImage (915) procedure prototype.

```
putpixelproc = procedure(X: SmallInt; Y: SmallInt; Color: ColorType
)
```

Standard pixel drawing routine prototype.

```
restorestateproc = procedure
```

Standard routine prototype to restore the graphical state at a closegraph call.

```
savestateproc = procedure
```

Standard routine prototype to save the graphical state before a mode is set.

```
setactivepageproc = procedure (page: Word)
```

Standard SetActivePage (915) procedure prototype.

```
SetAllPaletteProc = procedure (const Palette: PaletteType)
```

This routine prototype is a hook for SetAllPalette (915).

```
SetBkColorProc = procedure (ColorNum: ColorType)
```

SetBkColorProc is the procedure prototype for the SetBkColor (915) method handler in TMod-Info (937). The procedure gets passed the color code for the color to set as background color.

```
setrgbpaletteproc = procedure (ColorNum: SmallInt; RedValue: SmallInt
;
                                GreenValue: SmallInt; BlueValue:
                                SmallInt)
```

This routine prototype is a hook for SetRGBPalette (916).

```
setvisualpageproc = procedure (page: Word)
```

Standard SetVisualPage (916) procedure prototype.

```
TCharsetTransTable = Array[Char] of Char
```

Character transliteration table, with entries for 256 characters.

```
vlineproc = procedure (x: SmallInt; y: SmallInt; y2: SmallInt)
```

Standard procedure prototype to draw a single vertical line.

56.13.3 Variables

```
Circle : CircleProc
```

Circle draws a complete circle with center at (X, Y), radius radius.

```
ClearViewPort : clrviewproc
```

Clears the current viewport. The current background color is used as filling color. The pointer is set at (0, 0).

DirectPutPixel : defpixelproc

Hook to directly draw a pixel on the screen.

GetBkColor : GetBkColorProc

GetBkColor returns the current background color (the palette entry).

GetImage : getimageproc

GetImage Places a copy of the screen area (X1, Y1) to X2, Y2 in BitMap

GetPixel : getpixelproc

GetPixel returns the color of the point at (X, Y)

GetRGBPalette : getrgbpaletteproc

Hook to set a RGB palette entries.

GetScanLine : getscanlineproc

Hook to get a scan line from the screen.

GraphFreeMemPtr : graphfreememprc

Hook to free heap memory.

GraphGetMemPtr : graphgetmemprc

Hook to get heap memory.

HLine : hlineproc

Hook to draw a solid horizontal line.

ImageSize : imagesizeproc

ImageSize returns the number of bytes needed to store the image in the rectangle defined by (X1, Y1) and (X2, Y2).

InternalEllipse : ellipseproc

Hook to draw an ellipse.

Line : lineproc

`Line` draws a line starting from `(X1, Y1)` to `(X2, Y2)`, in the current line style and color. The current position is put to `(X2, Y2)`

`OutTextXY` : `OutTextXYProc`

`OutText` puts `TextString` on the screen, at position `(X, Y)`, using the current font and text settings. The current position is moved to the end of the text.

`PatternLine` : `patternlineproc`

Hook to draw a patterned line.

`PutImage` : `putimageproc`

`PutImage` Places the bitmap in `Bitmap` on the screen at `(X1, Y1)`. `How` determines how the bitmap will be placed on the screen. Possible values are:

- `CopyPut`
- `XORPut`
- `ORPut`
- `AndPut`
- `NotPut`

`PutPixel` : `putpixelproc`

Puts a point at `(X, Y)` using color `Color`

`RestoreVideoState` : `restorestateproc`

Hook to restore a saved video mode.

`SaveVideoState` : `savestateproc`

Hook to save the current video state.

`SetActivePage` : `setactivepageproc`

Sets `Page` as the active page for all graphical output.

`SetAllPalette` : `SetAllPaletteProc`

Sets the current palette to `Palette`. `Palette` is an untyped variable, usually pointing to a record of type `PaletteType`

`SetBkColor` : `SetBkColorProc`

Sets the background color to `Color`.

`SetRGBPalette` : `setrgbpaletteproc`

SetRGBPalette sets the ColorNr-th entry in the palette to the color with RGB-values Red, Green Blue.

SetVisualPage : setvisualpageproc

SetVisualPage sets the video page to page number Page.

VLine : vlineproc

Hook to draw a solid vertical line.

56.14 Procedures and functions

56.14.1 Arc

Synopsis: Draw part of a circle.

Declaration: `procedure Arc(X: SmallInt; Y: SmallInt; StAngle: Word; EndAngle: Word; Radius: Word)`

Visibility: default

Description: Arc draws part of a circle with center at (X,Y), radius radius, starting from angle start, stopping at angle stop. These angles are measured counterclockwise.

Errors: None.

See also: Circle (913), Ellipse (918), GetArcCoords (919), PieSlice (928), Sector (929)

56.14.2 Bar

Synopsis: Draw filled rectangle.

Declaration: `procedure Bar(x1: SmallInt; y1: SmallInt; x2: SmallInt; y2: SmallInt)`

Visibility: default

Description: Draws a rectangle with corners at (X1,Y1) and (X2,Y2) and fills it with the current color and fill-style.

Errors: None.

See also: Bar3D (916), Rectangle (928)

56.14.3 Bar3D

Synopsis: Draw filled 3-dimensional rectangle.

Declaration: `procedure Bar3D(x1: SmallInt; y1: SmallInt; x2: SmallInt; y2: SmallInt; depth: Word; top: Boolean)`

Visibility: default

Description: Bar3d draws a 3-dimensional Bar with corners at (X1,Y1) and (X2,Y2) and fills it with the current color and fill-style. Depth specifies the number of pixels used to show the depth of the bar.

If Top is true; then a 3-dimensional top is drawn.

Errors: None.

See also: Bar (916), Rectangle (928)

56.14.4 ClearDevice

Synopsis: Clear the complete screen.

Declaration: `procedure ClearDevice`

Visibility: `default`

Description: Clears the graphical screen (with the current background color), and sets the pointer at `(0, 0)`.

Errors: None.

See also: `ClearViewPort` (914), `SetBkColor` (915)

56.14.5 Closegraph

Synopsis: Close graphical system.

Declaration: `procedure Closegraph`

Visibility: `default`

Description: Closes the graphical system, and restores the screen modus which was active before the graphical modus was activated.

Errors: None.

See also: `InitGraph` (925)

56.14.6 DetectGraph

Synopsis: Detect correct graphical driver to use.

Declaration: `procedure DetectGraph(var GraphDriver: SmallInt;
var GraphMode: SmallInt)`

Visibility: `default`

Description: `DetectGraph` checks the hardware in the PC and determines the driver and screen-modus to be used. These are returned in `Driver` and `Modus`, and can be fed to `InitGraph`. See the `InitGraph` for a list of drivers and modi.

Errors: None.

See also: `InitGraph` (925)

56.14.7 DrawPoly

Synopsis: Draw a polygon.

Declaration: `procedure DrawPoly(NumPoints: Word; var polypoints)`

Visibility: `default`

Description: `DrawPoly` draws a polygon with `NumberOfPoints` corner points, using the current color and line-style. `PolyPoints` is an array of type `PointType` (936).

Errors: None.

See also: `Bar` (916), `Bar3D` (916), `Rectangle` (928)

56.14.8 Ellipse

Synopsis: Draw an ellipse.

Declaration: `procedure Ellipse(X: SmallInt; Y: SmallInt; stAngle: Word;
EndAngle: Word; XRadius: Word; YRadius: Word)`

Visibility: default

Description: `Ellipse` draws part of an ellipse with center at (X, Y) . `XRadius` and `Yradius` are the horizontal and vertical radii of the ellipse. `Start` and `Stop` are the starting and stopping angles of the part of the ellipse. They are measured counterclockwise from the X-axis (3 o'clock is equal to 0 degrees). Only positive angles can be specified.

Errors: None.

See also: [Arc \(916\)](#), [Circle \(913\)](#), [FillEllipse \(918\)](#)

56.14.9 FillEllipse

Synopsis: Draw and fill an ellipse.

Declaration: `procedure FillEllipse(X: SmallInt; Y: SmallInt; XRadius: Word;
YRadius: Word)`

Visibility: default

Description: `Ellipse` draws an ellipse with center at (X, Y) . `XRadius` and `Yradius` are the horizontal and vertical radii of the ellipse. The ellipse is filled with the current color and fill-style.

Errors: None.

See also: [Arc \(916\)](#), [Circle \(913\)](#), [GetArcCoords \(919\)](#), [PieSlice \(928\)](#), [Sector \(929\)](#)

56.14.10 FillPoly

Synopsis: Draw, close and fill a polygon.

Declaration: `procedure FillPoly(NumPoints: Word; var PolyPoints)`

Visibility: default

Description: `FillPoly` draws a polygon with `NumberOfPoints` corner points and fills it using the current color and line-style. `PolyPoints` is an array of type `PointType`.

Errors: None.

See also: [Bar \(916\)](#), [Bar3D \(916\)](#), [Rectangle \(928\)](#)

56.14.11 FloodFill

Synopsis: Fill an area with a given color.

Declaration: `procedure FloodFill(x: SmallInt; y: SmallInt; Border: ColorType)`

Visibility: default

Description: Fills the area containing the point (X, Y) , bounded by the color `BorderColor`.

Errors: None

See also: [SetColor \(930\)](#), [SetBkColor \(915\)](#)

56.14.12 GetArcCoords

Synopsis: Return coordinates of last drawn arc or ellipse.

Declaration: `procedure GetArcCoords (var ArcCoords: ArcCoordsType)`

Visibility: default

Description: `GetArcCoords` returns the coordinates of the latest `Arc` or `Ellipse` call.

Errors: None.

See also: `Arc` ([916](#)), `Ellipse` ([918](#))

56.14.13 GetAspectRatio

Synopsis: Return screen resolution.

Declaration: `procedure GetAspectRatio (var Xasp: Word; var Yasp: Word)`

Visibility: default

Description: `GetAspectRatio` determines the effective resolution of the screen. The aspect ration can then be calculated as `Xasp/Yasp`.

Errors: None.

See also: `InitGraph` ([925](#)), `SetAspectRatio` ([929](#))

56.14.14 GetColor

Synopsis: Return current drawing color.

Declaration: `function GetColor : ColorType`

Visibility: default

Description: `GetColor` returns the current drawing color (the palette entry).

Errors: None.

See also: `GetColor` ([919](#)), `SetBkColor` ([915](#))

56.14.15 GetDefaultPalette

Synopsis: Return default palette.

Declaration: `procedure GetDefaultPalette (var Palette: PaletteType)`

Visibility: default

Description: `GetDefaultPalette` returns the current palette in `Palette`.

Errors: None.

See also: `GetColor` ([919](#)), `GetBkColor` ([914](#))

56.14.16 GetDirectVideo

Synopsis: Determine whether direct video mode is active.

Declaration: `function GetDirectVideo : Boolean`

Visibility: default

Description: Determine whether direct video mode is active.

56.14.17 GetDriverName

Synopsis: Return current driver name.

Declaration: `function GetDriverName : string`

Visibility: default

Description: `GetDriverName` returns a string containing the name of the current driver.

Errors: None.

See also: `GetModeName` ([922](#)), `InitGraph` ([925](#))

56.14.18 GetFillPattern

Synopsis: Return current fill pattern.

Declaration: `procedure GetFillPattern(var FillPattern: FillPatternType)`

Visibility: default

Description: `GetFillPattern` returns an array with the current fill-pattern in `FillPattern`

Errors: None

See also: `SetFillPattern` ([930](#))

56.14.19 GetFillSettings

Synopsis: Return current fill settings.

Declaration: `procedure GetFillSettings(var Fillinfo: FillSettingsType)`

Visibility: default

Description: `GetFillSettings` returns the current fill-settings in `FillInfo`

Errors: None.

See also: `SetFillPattern` ([930](#))

56.14.20 GetGraphMode

Synopsis: Get current graphical modulus.

Declaration: `function GetGraphMode : SmallInt`

Visibility: default

Description: `GetGraphMode` returns the current graphical modulus

Errors: None.

See also: `InitGraph` ([925](#))

56.14.21 GetLineSettings

Synopsis: Get current line drawing settings.

Declaration: `procedure GetLineSettings (var ActiveLineInfo: LineSettingsType)`

Visibility: default

Description: `GetLineSettings` returns the current Line settings in `LineInfo`

Errors: None.

See also: `SetLineStyle` ([931](#))

56.14.22 GetMaxColor

Synopsis: return maximum number of colors.

Declaration: `function GetMaxColor : ColorType`

Visibility: default

Description: `GetMaxColor` returns the maximum color-number which can be set with `SetColor`. Contrary to Turbo Pascal, this color isn't always guaranteed to be white (for instance in 256+ color modes).

Errors: None.

See also: `SetColor` ([930](#)), `GetPaletteSize` ([923](#))

56.14.23 GetMaxMode

Synopsis: Return biggest mode for the current driver.

Declaration: `function GetMaxMode : SmallInt`

Visibility: default

Description: `GetMaxMode` returns the highest modulus for the current driver.

Errors: None.

See also: `InitGraph` ([925](#))

56.14.24 GetMaxX

Synopsis: Return maximal X coordinate.

Declaration: `function GetMaxX : SmallInt`

Visibility: default

Description: `GetMaxX` returns the maximum horizontal screen length

Errors: None.

See also: `GetMaxY` ([922](#))

56.14.25 GetMaxY

Synopsis: Return maximal Y coordinate.

Declaration: `function GetMaxY : SmallInt`

Visibility: default

Description: `GetMaxY` returns the maximum number of screen lines

Errors: None.

See also: `GetMaxX` ([922](#))

56.14.26 GetModeName

Synopsis: Return description a modus.

Declaration: `function GetModeName (ModeNumber: SmallInt) : string`

Visibility: default

Description: `GetModeName` Returns a string with the name of modus `Modus`

Errors: None.

See also: `GetDriverName` ([920](#)), `InitGraph` ([925](#))

56.14.27 GetModeRange

Synopsis: Return lowest and highest modus of current driver.

Declaration: `procedure GetModeRange (GraphDriver: SmallInt; var LoMode: SmallInt;
var HiMode: SmallInt)`

Visibility: default

Description: `GetModeRange` returns the Lowest and Highest modus of the currently installed driver. If no modes are supported for this driver, `HiModus` will be -1.

Errors: None.

See also: `InitGraph` ([925](#))

56.14.28 GetPalette

Synopsis: Return current palette.

Declaration: `procedure GetPalette (var Palette: PaletteType)`

Visibility: default

Description: `GetPalette` returns in `Palette` the current palette.

Errors: None.

See also: `GetPaletteSize` ([923](#)), `SetPalette` ([932](#))

56.14.29 GetPaletteSize

Synopsis: Return maximal number of entries in current palette.

Declaration: `function GetPaletteSize : SmallInt`

Visibility: default

Description: `GetPaletteSize` returns the maximum number of entries in the current palette.

Errors: None.

See also: `GetPalette` ([923](#)), `SetPalette` ([932](#))

56.14.30 GetTextSettings

Synopsis: Return current text style.

Declaration: `procedure GetTextSettings (var TextInfo: TextSettingsType)`

Visibility: default

Description: `GetTextSettings` returns the current text style settings : The font, direction, size and placement as set with `SetTextStyle` and `SetTextJustify`

Errors: None.

See also: `SetTextStyle` ([932](#)), `SetTextJustify` ([932](#))

56.14.31 GetViewSettings

Synopsis: Return current viewport.

Declaration: `procedure GetViewSettings (var viewport: ViewPortType)`

Visibility: default

Description: `GetViewSettings` returns the current viewport and clipping settings in `ViewPort`.

Errors: None.

See also: `SetViewPort` ([933](#))

56.14.32 GetX

Synopsis: Return current cursor X position.

Declaration: `function GetX : SmallInt`

Visibility: default

Description: `GetX` returns the X-coordinate of the current position of the graphical pointer

Errors: None.

See also: `GetY` ([924](#))

56.14.33 GetY

Synopsis: Return current cursor Y position.

Declaration: `function GetY : SmallInt`

Visibility: default

Description: `GetY` returns the Y-coordinate of the current position of the graphical pointer

Errors: None.

See also: `GetX` ([924](#))

56.14.34 GraphDefaults

Synopsis: Reset graphical mode to defaults.

Declaration: `procedure GraphDefaults`

Visibility: default

Description: `GraphDefaults` resets all settings for viewport, palette, foreground and background pattern, line-style and pattern, filling style, filling color and pattern, font, text-placement and text size.

Errors: None.

See also: `SetViewPort` ([933](#)), `SetFillStyle` ([930](#)), `SetColor` ([930](#)), `SetBkColor` ([915](#)), `SetLineStyle` ([931](#))

56.14.35 GraphErrorMsg

Synopsis: Return a description of an error.

Declaration: `function GraphErrorMsg(ErrorCode: SmallInt) : string`

Visibility: default

Description: `GraphErrorMsg` returns a string describing the error `Errorcode`. This string can be used to let the user know what went wrong.

Errors: None.

See also: `GraphResult` ([925](#))

56.14.36 GraphResult

Synopsis: Result of last graphical operation.

Declaration: `function GraphResult : SmallInt`

Visibility: default

Description: `GraphResult` returns an error-code for the last graphical operation. If the returned value is zero, all went well. A value different from zero means an error has occurred. besides all operations which draw something on the screen, the following procedures also can produce a `GraphResult` different from zero:

- `InstallUserFont` ([926](#))
- `SetLineStyle` ([931](#))
- `SetWriteMode` ([934](#))
- `SetFillStyle` ([930](#))
- `SetTextJustify` ([932](#))
- `SetGraphMode` ([931](#))
- `SetTextStyle` ([932](#))

Errors: None.

See also: `GraphErrorMsg` ([924](#))

56.14.37 InitGraph

Synopsis: Initialize graphical system.

Declaration: `procedure InitGraph(var GraphDriver: SmallInt; var GraphMode: SmallInt; const PathToDriver: string)`

Visibility: default

Description: `InitGraph` initializes the graph package. `GraphDriver` has two valid values: `GraphDriver=0` which performs an auto detect and initializes the highest possible mode with the most colors. 1024x768x64K is the highest possible resolution supported by the driver, if you need a higher resolution, you must edit `MODES.PPI`. If you need another mode, then set `GraphDriver` to a value different from zero and `graphmode` to the mode you wish (VESA modes where 640x480x256 is 101h etc.). `PathToDriver` is only needed, if you use the BGI fonts from Borland. Free Pascal does not offer BGI fonts like Borland, these must be obtained separately.

Example code:

```
var
  gd,gm : integer;
  PathToDriver : string;
begin
  gd:=detect; { highest possible resolution }
  gm:=0; { not needed, auto detection }
  PathToDriver:='C:\PP\BGI'; { path to BGI fonts,
                             drivers aren't needed }

  InitGraph(gd,gm,PathToDriver);
  if GraphResult<>grok then
    halt; ..... { whatever you need }
  CloseGraph; { restores the old graphics mode }
end.
```

Errors: None.

See also: Modes ([888](#)), DetectGraph ([917](#)), CloseGraph ([917](#)), GraphResult ([925](#))

56.14.38 InstallUserDriver

Synopsis: Install a user driver.

Declaration: `function InstallUserDriver (Name: string; AutoDetectPtr: Pointer)
: SmallInt`

Visibility: default

Description: `InstallUserDriver` adds the device-driver `DriverPath` to the list of .BGI drivers. `AutoDetectPtr` is a pointer to a possible auto-detect function.

Errors: None.

See also: `InitGraph` ([925](#)), `InstallUserFont` ([926](#))

56.14.39 InstallUserFont

Synopsis: Install a user-defined font.

Declaration: `function InstallUserFont (const FontFileName: string) : SmallInt`

Visibility: default

Description: `InstallUserFont` adds the font in `FontPath` to the list of fonts of the .BGI system.

Errors: None.

See also: `InitGraph` ([925](#)), `InstallUserDriver` ([926](#))

56.14.40 LineRel

Synopsis: Draw a line starting from current position in given direction.

Declaration: `procedure LineRel (Dx: SmallInt; Dy: SmallInt)`

Visibility: default

Description: `LineRel` draws a line starting from the current pointer position to the point (DX, DY) , **relative** to the current position, in the current line style and color. The Current Position is set to the endpoint of the line.

Errors: None.

See also: `Line` ([915](#)), `LineTo` ([927](#))

56.14.41 LineTo

Synopsis: Draw a line starting from current position to a given point.

Declaration: `procedure LineTo(X: SmallInt; Y: SmallInt)`

Visibility: default

Description: `LineTo` draws a line starting from the current pointer position to the point (DX, DY), **relative** to the current position, in the current line style and color. The Current position is set to the end of the line.

Errors: None.

See also: `LineRel` ([926](#)), `Line` ([915](#))

56.14.42 MoveRel

Synopsis: Move cursor relative to current position.

Declaration: `procedure MoveRel(Dx: SmallInt; Dy: SmallInt)`

Visibility: default

Description: `MoveRel` moves the pointer to the point (DX, DY), relative to the current pointer position

Errors: None.

See also: `MoveTo` ([927](#))

56.14.43 MoveTo

Synopsis: Move cursor to absolute position.

Declaration: `procedure MoveTo(X: SmallInt; Y: SmallInt)`

Visibility: default

Description: `MoveTo` moves the pointer to the point (X, Y).

Errors: None.

See also: `MoveRel` ([927](#))

56.14.44 OutText

Synopsis: Write text on the screen at the current location.

Declaration: `procedure OutText(const TextString: string)`

Visibility: default

Description: `OutText` puts `TextString` on the screen, at the current pointer position, using the current font and text settings. The current position is moved to the end of the text.

Errors: None.

See also: `OutTextXY` ([915](#))

56.14.45 PieSlice

Synopsis: Draw a pie-slice.

Declaration: `procedure PieSlice(X: SmallInt; Y: SmallInt; stangle: SmallInt;
 endAngle: SmallInt; Radius: Word)`

Visibility: default

Description: `PieSlice` draws and fills a sector of a circle with center (X, Y) and radius Radius, starting at angle Start and ending at angle Stop.

Errors: None.

See also: Arc ([916](#)), Circle ([913](#)), Sector ([929](#))

56.14.46 queryadapterinfo

Synopsis: Function called to retrieve the current video adapter settings.

Declaration: `function queryadapterinfo : PModeInfo`

Visibility: default

56.14.47 Rectangle

Synopsis: Draw a rectangle on the screen.

Declaration: `procedure Rectangle(x1: SmallInt; y1: SmallInt; x2: SmallInt;
 y2: SmallInt)`

Visibility: default

Description: Draws a rectangle with corners at (X1, Y1) and (X2, Y2), using the current color and style.

Errors: None.

See also: Bar ([916](#)), Bar3D ([916](#))

56.14.48 RegisterBGIDriver

Synopsis: Register a new BGI driver.

Declaration: `function RegisterBGIDriver(driver: pointer) : SmallInt`

Visibility: default

Description: Registers a user-defined BGI driver

Errors: None.

See also: `InstallUserDriver` ([926](#)), `RegisterBGIFont` ([929](#))

56.14.49 RegisterBGIfont

Synopsis: Register a new BGI font.

Declaration: `function RegisterBGIfont (font: pointer) : SmallInt`

Visibility: default

Description: Registers a user-defined BGI driver

Errors: None.

See also: [InstallUserFont \(926\)](#), [RegisterBGIDriver \(928\)](#)

56.14.50 RestoreCrtMode

Synopsis: Restore text screen.

Declaration: `procedure RestoreCrtMode`

Visibility: default

Description: Restores the screen modus which was active before the graphical modus was started.

To get back to the graph mode you were last in, you can use `SetGraphMode (GetGraphMode)`

Errors: None.

See also: [InitGraph \(925\)](#)

56.14.51 Sector

Synopsis: Draw and fill a sector of an ellipse.

Declaration: `procedure Sector (x: SmallInt; y: SmallInt; StAngle: Word;
EndAngle: Word; XRadius: Word; YRadius: Word)`

Visibility: default

Description: `Sector` draws and fills a sector of an ellipse with center (X, Y) and radii XRadius and YRadius, starting at angle Start and ending at angle Stop.

Errors: None.

See also: [Arc \(916\)](#), [Circle \(913\)](#), [PieSlice \(928\)](#)

56.14.52 SetAspectRatio

Synopsis: Set aspect ration of the screen.

Declaration: `procedure SetAspectRatio (Xasp: Word; Yasp: Word)`

Visibility: default

Description: Sets the aspect ratio of the current screen to Xasp/Yasp.

Errors: None

See also: [InitGraph \(925\)](#), [GetAspectRatio \(919\)](#)

56.14.53 SetColor

Synopsis: Set foreground drawing color.

Declaration: `procedure SetColor (Color: ColorType)`

Visibility: default

Description: Sets the foreground color to `Color`.

Errors: None.

See also: [GetColor \(919\)](#), [SetBkColor \(915\)](#), [SetWriteMode \(934\)](#)

56.14.54 SetDirectVideo

Synopsis: Attempt to enter direct video mode.

Declaration: `procedure SetDirectVideo (DirectAccess: Boolean)`

Visibility: default

Description: `SetDirectVideo` attempts to enter direct video mode. In that mode, everything is drawn straight in the video buffer.

56.14.55 SetFillPattern

Synopsis: Set drawing fill pattern.

Declaration: `procedure SetFillPattern (Pattern: FillPatternType; Color: ColorType)`

Visibility: default

Description: `SetFillPattern` sets the current fill-pattern to `FillPattern`, and the filling color to `Color`. The pattern is an 8x8 raster, corresponding to the 64 bits in `FillPattern`.

Errors: None

See also: [GetFillPattern \(920\)](#), [SetFillStyle \(930\)](#), [SetWriteMode \(934\)](#)

56.14.56 SetFillStyle

Synopsis: Set drawing fill style.

Declaration: `procedure SetFillStyle (Pattern: Word; Color: ColorType)`

Visibility: default

Description: `SetFillStyle` sets the filling pattern and color to one of the predefined filling patterns. `Pattern` can be one of the following predefined constants :

EmptyFill Uses backgroundcolor.

SolidFill Uses filling color

LineFill Fills with horizontal lines.

ltSlashFill Fills with lines from left-under to top-right.

SlashFill Idem as previous, thick lines.

BkSlashFill Fills with thick lines from left-Top to bottom-right.

LtBkSlashFill Idem as previous, normal lines.

HatchFill Fills with a hatch-like pattern.

XHatchFill Fills with a hatch pattern, rotated 45 degrees.

InterLeaveFill

WideDotFill Fills with dots, wide spacing.

CloseDotFill Fills with dots, narrow spacing.

UserFill Fills with a user-defined pattern.

Errors: None.

See also: [SetFillPattern \(930\)](#), [SetWriteMode \(934\)](#)

56.14.57 SetGraphMode

Synopsis: Set graphical mode.

Declaration: `procedure SetGraphMode (Mode: SmallInt)`

Visibility: default

Description: `SetGraphMode` sets the graphical mode and clears the screen.

Errors: None.

See also: [InitGraph \(925\)](#)

56.14.58 SetLineStyle

Synopsis: Set line drawing style.

Declaration: `procedure SetLineStyle (LineStyle: Word; Pattern: Word; Thickness: Word)`

Visibility: default

Description: `SetLineStyle` sets the drawing style for lines. You can specify a `LineStyle` which is one of the following predefined constants:

SolidLn draws a solid line.

DottedLn Draws a dotted line.

CenterLn draws a non-broken centered line.

DashedLn draws a dashed line.

UserBitLn Draws a User-defined bit pattern.

If `UserBitLn` is specified then `Pattern` contains the bit pattern. In all another cases, `Pattern` is ignored. The parameter `Width` indicates how thick the line should be. You can specify one of the following predefined constants:

NormWidth Normal line width

ThickWidth Double line width

Errors: None.

See also: [GetLineSettings \(921\)](#), [SetWriteMode \(934\)](#)

56.14.59 SetPalette

Synopsis: Set palette entry using color constant.

Declaration: `procedure SetPalette (ColorNum: Word; Color: ShortInt)`

Visibility: default

Description: `SetPalette` changes the `ColorNr`-th entry in the palette to `NewColor`

Errors: None.

See also: `SetAllPalette` (915), `SetRGBPalette` (916)

56.14.60 SetTextJustify

Synopsis: Set text placement style.

Declaration: `procedure SetTextJustify (horiz: Word; vert: Word)`

Visibility: default

Description: `SetTextJustify` controls the placement of new text, relative to the (graphical) cursor position. `Horizontal` controls horizontal placement, and can be one of the following predefined constants:

LeftText Text is set left of the pointer.

CenterText Text is set centered horizontally on the pointer.

RightText Text is set to the right of the pointer.

`Vertical` controls the vertical placement of the text, relative to the (graphical) cursor position. Its value can be one of the following predefined constants :

BottomText Text is placed under the pointer.

CenterText Text is placed centered vertically on the pointer.

TopText Text is placed above the pointer.

Errors: None.

See also: `OutText` (927), `OutTextXY` (915)

56.14.61 SetTextStyle

Synopsis: Set text style.

Declaration: `procedure SetTextStyle (font: Word; direction: Word; charsize: Word)`

Visibility: default

Description: `SetTextStyle` controls the style of text to be put on the screen. predefined constants for `Font` are:

DefaultFont The default font

TriplexFont A special font

SmallFont A smaller font

SansSerifFont A sans-serif font (like Arial)

GothicFont A gothic font

ScriptFont A script font

SimpleFont A simple font

TSCRFont Terminal screen font

LCOMFont ?

EuroFont ?

BoldFont A bold typeface font

predefined constants for `Direction` are :

HorizDir Write horizontal

VertDir Write vertical

Errors: None.

See also: `GetTextSettings` ([923](#))

56.14.62 SetUserCharSize

Synopsis: Set user character size for vector font.

Declaration: `procedure SetUserCharSize (Multx: Word; Divx: Word; Multy: Word;
Divy: Word)`

Visibility: default

Description: Sets the width and height of vector-fonts. The horizontal size is given by `Xasp1/Xasp2`, and the vertical size by `Yasp1/Yasp2`.

Errors: None.

See also: `SetTextStyle` ([932](#))

56.14.63 SetViewPort

Synopsis: Set the graphical drawing window.

Declaration: `procedure SetViewPort (X1: SmallInt; Y1: SmallInt; X2: SmallInt;
Y2: SmallInt; Clip: Boolean)`

Visibility: default

Description: Sets the current graphical viewport (window) to the rectangle defined by the top-left corner `(X1, Y1)` and the bottom-right corner `(X2, Y2)`. If `Clip` is true, anything drawn outside the viewport (window) will be clipped (i.e. not drawn). Coordinates specified after this call are relative to the top-left corner of the viewport.

Errors: None.

See also: `GetViewSettings` ([923](#))

56.14.64 SetWriteMode

Synopsis: Specify binary operation to perform when drawing on screen.

Declaration: `procedure SetWriteMode (WriteMode: SmallInt)`

Visibility: default

Description: `SetWriteMode` controls the drawing of lines on the screen. It controls the binary operation used when drawing lines on the screen. Mode can be one of the following predefined constants:

CopyPut Draw as specified using current bitmask and color

XORPut Draw XOR-ing current bitmask and color

Errors: None.

See also: `SetColor` (930), `SetBkColor` (915), `SetLineStyle` (931), `SetFillStyle` (930)

56.14.65 SetWriteModeEx

Synopsis: Set write mode (extended version).

Declaration: `procedure SetWriteModeEx (WriteMode: SmallInt)`

Visibility: default

Description: `SetWriteModeEx` sets the graph write mode similar to `SetWriteMode` (934) but only if the `WriteMode` is in the range `CopyPut..NotPut`. If the provided value is outside this range, it is not set.

See also: `SetWriteMode` (934)

56.14.66 TextHeight

Synopsis: Return height (in pixels) of the given string.

Declaration: `function TextHeight (const TextString: string) : Word`

Visibility: default

Description: `TextHeight` returns the height (in pixels) of the string `S` in the current font and text-size.

Errors: None.

See also: `TextWidth` (934)

56.14.67 TextWidth

Synopsis: Return width (in pixels) of the given string.

Declaration: `function TextWidth (const TextString: string) : Word`

Visibility: default

Description: `TextWidth` returns the width (in pixels) of the string `S` in the current font and text-size.

Errors: None.

See also: `TextHeight` (934)

56.15 ArcCoordsType

```
ArcCoordsType = record
  x : SmallInt;
  y : SmallInt;
  xstart :
    SmallInt;
  ystart : SmallInt;
  xend : SmallInt;
  yend : SmallInt
  ;
end
```

Describe the last arc which was drawn on screen.

56.16 FillSettingsType

```
FillSettingsType = record
  pattern : Word;
  color : ColorType;
end
```

Record describing fill mode.

56.17 LineSettingsType

```
LineSettingsType = record
  linestyle : Word;
  pattern : Word;
  thickness : Word;
end
```

Record describing current line drawing mode.

56.18 PaletteType

```
PaletteType = record
  Size : LongInt;
  Colors : Array[0..MaxColors
  ] of RGBRec;
end
```

Record describing palette.

56.19 PointType

```
PointType = record
```



```
x : SmallInt;  
y : SmallInt;  
end
```

Record describing a point in a 2 dimensional plane.

56.20 RGBRec

```
RGBRec = packed record  
  Red : SmallInt;  
  Green : SmallInt;  
  Blue  
  : SmallInt;  
end
```

Record describing palette RGB color.

56.21 TextSettingsType

```
TextSettingsType = record  
  font : Word;  
  direction : Word;  
  charsize  
  : Word;  
  horiz : Word;  
  vert : Word;  
end
```

Record describing how texts are drawn.

56.22 TModeInfo

```
TModeInfo = record  
  DriverNumber : SmallInt;  
  ModeNumber : SmallInt  
  ;  
  internModeNumber : SmallInt;  
  MaxColor : LongInt;  
  PaletteSize  
  : LongInt;  
  XAspect : Word;  
  YAspect : Word;  
  MaxX : Word;  
  MaxY  
  : Word;  
  DirectColor : Boolean;  
  Hardwarepages : Byte;  
  ModeName
```

```

: string;
DirectPutPixel : defpixelproc;
GetPixel : getpixelproc
;
PutPixel : putpixelproc;
SetRGBPalette : setrgbpaletteproc;
GetRGBPalette : getrgbpaletteproc;
SetAllPalette : SetAllPaletteProc
;
SetVisualPage : setvisualpageproc;
SetActivePage : setactivepageproc
;
ClearViewPort : clrviewproc;
PutImage : putimageproc;
GetImage
: getimageproc;
ImageSize : imagesizeproc;
GetScanLine : getscanlineproc
;
Line : lineproc;
InternalEllipse : ellipseproc;
PatternLine
: patternlineproc;
HLine : hlineproc;
VLine : vlineproc;
Circle
: CircleProc;
InitMode : initmodeproc;
OutTextXY : OutTextXYProc
;
SetBKColor : SetBkColorProc;
GetBKColor : GetBkColorProc;
next : PModeInfo;
end

```

Record describing a graphical mode.

56.23 TResolutionRec

```

TResolutionRec = record
  x : LongInt;
  y : LongInt;
end

```

Record describing resolution.

56.24 ViewPortType

```

ViewPortType = record
  x1 : SmallInt;

```

```
y1 : SmallInt;  
x2 : SmallInt  
;  
y2 : SmallInt;  
Clip : Boolean;  
end
```

Record describing a viewport.

Chapter 57

Reference for unit 'heaptrc'

57.1 Used units

Table 57.1: Used units by unit 'heaptrc'

Name	Page
System	1335

57.2 Overview

This document describes the HEAPTRC unit for Free Pascal. It was written by Pierre Muller. It is system independent, and works on all supported systems.

The HEAPTRC unit can be used to debug your memory allocation/deallocation. It keeps track of the calls to getmem/freemem, and, implicitly, of New/Dispose statements.

When the program exits, or when you request it explicitly. It displays the total memory used, and then dumps a list of blocks that were allocated but not freed. It also displays where the memory was allocated.

If there are any inconsistencies, such as memory blocks being allocated or freed twice, or a memory block that is released but with wrong size, this will be displayed also.

The information that is stored/displayed can be customized using some constants.

Do not use this unit directly, instead use the `-gh` switch, to let the compiler insert the unit in the uses clause. You can detect whether the heaptrc unit is actually used in the main program code with the following function:

```
function IsHeapTraceActive : boolean;

begin
  {$if declared(UseHeapTrace) }
    Result:=UseHeapTrace;
  {$else}
    Result:=False;
  {$endif}
end;
```

57.3 Controlling HeapTrc with environment variables.

The HeapTrc unit can be controlled with the HEAPTRC environment variable. The contents of this variable controls the initial setting of some constants in the unit. HEAPTRC consists of one or more of the following strings, separated by spaces:

keepreleased If this string occurs, then the KeepReleased (942) variable is set to `True`

disabled If this string occurs, then the UseHeapTrace (943) variable is set to `False` and the heap trace is disabled. It does not make sense to combine this value with other values.

nohalt If this string occurs, then the HaltOnError (941) variable is set to `False`, so the program continues executing even in case of a heap error.

log=filename If this string occurs, then the output of heaptrc is sent to the specified Filename. (see also SetHeapTraceOutput (945))

The following are valid values for the HEAPTRC variable:

```
HEAPTRC=disabled
HEAPTRC="keepreleased log=heap.log"
HEAPTRC="log=myheap.log nohalt"
```

Note that these strings are case sensitive, and the name of the variable too.

57.4 HeapTrc Usage.

You must use the `-gh` switch, to let the compiler insert the unit by itself, so you don't have to include it in your `uses` clause. In fact, as of version 3.0.0 you may no longer do so.

The below example shows how to use the heaptrc unit.

This is the memory dump shown when running this program in a standard way:

```
Marked memory at 0040FA50 invalid
Wrong size : 128 allocated 64 freed
  0x00408708
  0x0040CB49
  0x0040C481
Call trace for block 0x0040FA50 size 128
  0x0040CB3D
  0x0040C481
```

If you use the `lineinfo` unit (or use the `-gl` switch) as well, then heaptrc will also give you the filenames and line-numbers of the procedures in the backtrace:

```
Marked memory at 00410DA0 invalid
Wrong size : 128 allocated 64 freed
  0x004094B8
  0x0040D8F9  main,   line 25 of heapex.pp
  0x0040D231
Call trace for block 0x00410DA0 size 128
  0x0040D8ED  main,   line 23 of heapex.pp
  0x0040D231
```

If lines without filename/line-number occur, this means there is a unit which has no debug info included.

Listing: ./examples/heapex/heapex.pp

```

Program heapex;

{
  Program used to demonstrate the usage of heaptrc unit
  Compile this program with the -gh command-line option
}

Var P1 : ^Longint;
      P2 : Pointer;
      I : longint;

begin
  {$IF NOT DECLARED(heaptrc)}
  {$ERROR You must compile this program with -gh}
  {$ENDIF}
  New(P1);
  // causes previous allocation not to be de-allocated
  New(P1);
  Dispose(P1);
  For I:=1 to 10 do
    begin
      GetMem (P2,128);
      // When I is even, deallocate block. We loose 5 times 128
      // bytes this way.
      If (I mod 2) = 0 Then FreeMem(P2,128);
      end;
      GetMem(P2,128);
      // This will provoke an error and a memory dump
      Freemem (P2,64);
    end.
  end.

```

57.5 Constants, types and variables

57.5.1 Constants

```
add_tail : Boolean = True
```

If `add\_tail` is `True` (the default) then a check is also performed on the memory location just behind the allocated memory.

```
GlobalSkipIfNoLeaks : Boolean = False
```

`GlobalSkipIfNoLeaks` will, when set to `true`, disable the output of the `heaptrc` unit if no memory leaks will be detected, i.e. there will only be output if there were actual memory leaks.

```
HaltOnError : Boolean = True
```

If `HaltOnError` is set to `True` then an illegal call to `FreeMem` will cause the memory manager to execute a `halt (1)` instruction, causing a memory dump. By Default it is set to `True`.

```
HaltOnNotReleased : Boolean = False
```

`HaltOnNotReleased` can be set to `True` to make the `DumpHeap` (943) procedure halt (exit code 203) the program if any memory was not released when the dump is made. If it is `False` (the default) then `DumpHeap` just returns.

```
keepreleased : Boolean = False
```

If `keepreleased` is set to `true`, then a list of freed memory blocks is kept. This is useful if you suspect that the same memory block is released twice. However, this option is very memory intensive, so use it sparingly, and only when it's really necessary.

```
maxprintedblocklength : Integer = 128
```

`maxprintedblocklength` determines the maximum number of bytes written by a memory block dump, as produced when `printleakedblock` (942) or `printfaultyblock` (942) are true. If the size of the memory block is larger than this size, then only the first `maxprintedblocklength` will be included in the dump.

```
printfaultyblock : Boolean = False
```

`printleakedblock` can be set to `True` to print a memory dump of faulty memory blocks (in case a memory override occurs) The block is printed as a series of hexadecimal numbers, representing the bytes in the memory block. At most `maxprintedblocklength` (942) bytes of the memory block will be printed.

```
printleakedblock : Boolean = False
```

`printleakedblock` can be set to `True` to print a memory dump of unreleased blocks when the `heaptrc` unit produces a summary of memory leaks. The block is printed as a series of hexadecimal numbers, representing the bytes in the memory block. At most `maxprintedblocklength` (942) bytes of the memory block will be printed.

```
quicktrace : Boolean = True
```

`Quicktrace` determines whether the memory manager checks whether a block that is about to be released is allocated correctly. This is a rather time consuming search, and slows program execution significantly, so by default it is set to `True`.

```
tail_size : LongInt = sizeof(ptruint)
```

This is the size of the tail block added to every memory block when `add_tail` (941) is `True`.

```
tracesize = 16
```

`Tracesize` specifies how many levels of calls are displayed of the call stack during the memory dump. If you specify `keepreleased:=True` then half the `TraceSize` is reserved for the `GetMem` call stack, and the other half is reserved for the `FreeMem` call stack. For example, the default value of 8 will cause eight levels of call frames to be dumped for the `getmem` call if `keepreleased` is `False`. If `KeepReleased` is `true`, then 4 levels of call frames will be dumped for the `GetMem` call and 4 frames will be dumped for the `FreeMem` call. If you want to change this value, you must recode the `heaptrc` unit.

```
usecrc : Boolean = True
```

If `usecrc` is `True` (the default) then a crc check is performed on locations before and after the allocated memory. This is useful to detect memory overwrites.

```
useheaptrace : Boolean = True
```

This variable can be set to `False` at program startup to disable the heap trace functionality completely. Because this variable is checked in the initialization section of the unit, the only way to do this is through the help of an environment variable (940), which must be set before the program starts. It follows that setting the variable to `False` while the main program is executed will have no effect other than not displaying the heap usage report at program exit.

Note that this disables the heaptrace functionality completely: the tracing heapmanager is simply not installed if the variable is set to `False`.

57.5.2 Types

```
tDisplayExtraInfoProc = procedure(var ptext: text; p: pointer)
```

The `TDisplayExtraInfoType` is a procedural type used in the `SetHeapExtraInfo` (944) call to display a memory location which was previously filled with `TFillExtraInfoProc` (943)

```
tFillExtraInfoProc = procedure(p: pointer)
```

The `TFillExtraInfoProc` is a procedural type used in the `SetHeapExtraInfo` (944) call to fill a memory location with extra data for displaying.

57.6 Procedures and functions

57.6.1 CheckPointer

Synopsis: Check if a pointer is in the address range of the application.

Declaration: `procedure CheckPointer(p: pointer)`

Visibility: default

Description: `CheckPointer` checks if the pointer is in the address range of the application, more specifically, if it is in the heap. if not, it prints an error and stops the program with aruntime error 204.

57.6.2 DumpHeap

Synopsis: Dump memory usage report to stderr.

Declaration: `procedure DumpHeap`
`procedure DumpHeap(SkipIfNoLeaks: Boolean)`

Visibility: default

Description: `DumpHeap` dumps to standard error a summary of memory usage. It is called automatically by the heaptrc unit when your program exits (by installing an exit procedure), but it can be called at any time.

Errors: None.

57.6.3 SetHeapExtraInfo

Synopsis: Store extra information in blocks.

Declaration: `procedure SetHeapExtraInfo(size: PtrUInt; fillproc: tFillExtraInfoProc; displayproc: tdisplayextrainfoProc)`

Visibility: default

Description: You can use `SetHeapExtraInfo` to store extra info in the blocks that the `heaptrc` unit reserves when tracing `getmem` calls. `Size` indicates the size (in bytes) that the trace mechanism should reserve for your extra information. For each call to `getmem`, `FillProc` will be called, and passed a pointer to the memory reserved.

When dumping the memory summary, the extra info is shown by calling `displayproc` and passing it the memory location which was filled by `fillproc`. It should write the information in readable form to the text file provided in the call to `displayproc`.

Errors: You can only call `SetHeapExtraInfo` if no memory has been allocated yet. If memory was already allocated prior to the call to `SetHeapExtraInfo`, then an error will be displayed on standard error output, and a `DumpHeap` (943) is executed.

See also: `DumpHeap` (943), `SetHeapTraceOutput` (945)

Listing: `./examples/heapex/setinfo.pp`

Program `heapex`;

{ Program used to demonstrate the usage of heaptrc unit }

{ you must compile this program with the `-gh` command-line switch of FPC }

Var `P1 : ^Longint;`
 `P2 : Pointer;`
 `I : longint;`
 `Marker : Longint;`

Procedure `SetMarker (P : pointer);`

Type `PLongint = ^Longint;`

begin
 `PLongint(P)^:=Marker;`
end;

Procedure `Part1;`

begin
 // Blocks allocated here are marked with \$FFAAFFAA = -5570646
 `Marker := $FFAAFFAA;`
 `New(P1);`
 `New(P1);`
 `Dispose(P1);`
 For `I:=1 to 10 do`
 begin
 `GetMem (P2,128);`
 If `(I mod 2) = 0 Then FreeMem(P2,128);`
 end;
 `GetMem(P2,128);`
 end;

Procedure Part2;

```

begin
  // Blocks allocated here are marked with $FAFAFAFA = -84215046
  Marker := $FAFAFAFA;
  New(P1);
  New(P1);
  Dispose(P1);
  For I:=1 to 10 do
    begin
      GetMem (P2,128);
      If (I mod 2) = 0 Then FreeMem(P2,128);
    end;
    GetMem(P2,128);
  end;

  begin
    SetExtraInfo(SizeOf(Marker),@SetMarker);
    Writeln ('Part 1');
    part1;
    Writeln('Part 2');
    part2;
  end.

```

57.6.4 SetHeapTraceOutput

Synopsis: Specify filename or text file for heap trace output.

Declaration: `procedure SetHeapTraceOutput(const name: string); Overload`
`procedure SetHeapTraceOutput(var ATextOutput: Text); Overload`

Visibility: default

Description: `SetHeapTraceOutput` using the `name` argument sets the filename into which heap trace info will be written. You can also use the `ATextOutput` form, where you pass an initialized text file record, this record will then be used to write the heap trace.

By default information is written to standard error, this function allows you to redirect the information to a file with full filename `name` or a text file `ATextOutput`.

Errors: If the file cannot be written to, errors will occur when writing the trace.

See also: `SetHeapExtraInfo` ([944](#))

Chapter 58

Reference for unit 'ipc'

58.1 Used units

Table 58.1: Used units by unit 'ipc'

Name	Page
BaseUnix	155
System	1335
unixtype	2229

58.2 Overview

This document describes the IPC unit for Free Pascal. It was written for Linux by Michael Van Canneyt. It gives all the functionality of System V Inter-Process Communication: shared memory, semaphores and messages. It works only on the Linux operating system.

Many constants here are provided for completeness only, and should under normal circumstances not be used by the programmer.

58.3 Constants, types and variables

58.3.1 Constants

`IPC_CREAT = 1 shl 9`

Create if key is nonexistent.

`IPC_EXCL = 2 shl 9`

fail if key exists.

`IPC_INFO = 3`

For `ipcs` call.

IPC_NOWAIT = 4 shl 9

return error on wait.

IPC_PRIVATE = Tkey(0)

IPC_RMID = 0

Remove resource.

IPC_SET = 1

set ipc_perm options.

IPC_STAT = 2

get ipc_perm options.

MSGMAX = 4056

Internal Message control code. Do not use.

MSGMNB = 16384

Internal Message control code. Do not use.

MSGMNI = 128

Internal Message control code. Do not use.

MSG_EXCEPT = 2 shl 12

Internal Message control code. Do not use.

MSG_NOERROR = 1 shl 12

Internal Message control code. Do not use.

SEM_GETALL = 13

Semaphore operation: Get all semaphore values.

SEM_GETNCNT = 14

Semaphore operation: Get number of processes waiting for resource.

SEM_GETPID = 11

Semaphore operation: Get process ID of last operation.

SEM_GETVAL = 12

Semaphore operation: Get current value of semaphore.

SEM_GETZCNT = 15

Semaphore operation: Get number of processes waiting for semaphores to reach zero.

SEM_SEMMNI = 128

Semaphore operation: ?

SEM_SEMNS = SEM_SEMMNI * SEM_SEMMSL

Semaphore operation: ?

SEM_SEMMSL = 32

Semaphore operation: ?

SEM_SEMOPM = 32

Semaphore operation: ?

SEM_SEVMX = 32767

Semaphore operation: ?

SEM_SETALL = 17

Semaphore operation: Set all semaphore values.

SEM_SETVAL = 16

Semaphore operation: Set semaphore value.

SEM_UNDO = \$1000

Constant for use in semop (960).

SHM_LOCK = 11

This constant is used in the shmctl (962) call.

SHM_R = 4 shl 6

This constant is used in the shmctl (962) call.

SHM_RDONLY = 1 shl 12

This constant is used in the shmctl (962) call.

`SHM_REMAP = 4 shl 12`

This constant is used in the `shmctl` (962) call.

`SHM_RND = 2 shl 12`

This constant is used in the `shmctl` (962) call.

`SHM_UNLOCK = 12`

This constant is used in the `shmctl` (962) call.

`SHM_W = 2 shl 6`

This constant is used in the `shmctl` (962) call.

58.3.2 Types

`key_t = TKey`

Alias for `TKey` (950) type.

`msglen_t = culong`

Message length type.

`msgqnum_t = culong`

Message queue number type.

`PIPC_Perm = ^TIPC_Perm`

Pointer to `TIPC_Perm` (965) record.

`PMSG = ^TMSG`

Pointer to `TMSG` (965) record.

`PMSGbuf = ^TMSGbuf`

Pointer to `TMsgBuf` (965) record.

`PMSQid_ds = ^TMSQid_ds`

Pointer to `TMSQid_ds` (966).

`PSEMbuf = ^TSEMbuf`

Pointer to `TSembuf` (966) record.

```
PSEMid_ds = ^TSEMid_ds
```

Pointer to TSEMid_ds (967) record.

```
PSEMinfo = ^TSEMinfo
```

Pointer to TSEMinfo (967) record.

```
PSEMun = ^TSEMun
```

Pointer to TSEMun (950) record.

```
PShmid_DS = ^TShmid_ds
```

Pointer to TSHMid_ds (967) record.

```
PSHM_info = ^TSHM_info
```

```
TKey = cint
```

Type returned by the ftok (950) key generating function.

```
TSEMun = record
case cint of
0: (
    val : cint;
);
1: (
    buf : PSEMid_ds
    ;
);
2: (
    arr : PWord;
);
3: (
    padbuf : PSEMinfo;
);
4: (
    padpad
    : pointer;
);
end
```

Record used in semctl (955) call.

58.4 Procedures and functions

58.4.1 ftok

Synopsis: Create token from filename.

Declaration: `function ftok(Path: PChar; ID: cint) : TKey`

Visibility: `default`

Description: `ftok` returns a key that can be used in a `semget` (960), `shmget` (964) or `msgget` (953) call to access a new or existing IPC resource.

`Path` is the name of a file in the file system, `ID` is a character of your choice. The `ftok` call does the same as it's C counterpart, so a pascal program and a C program will access the same resource if they use the same `Path` and `ID`

For an example, see `msgctl` (951), `semctl` (955) or `shmctl` (962).

Errors: `ftok` returns -1 if the file in `Path` doesn't exist.

See also: `semget` (960), `shmget` (964), `msgget` (953)

58.4.2 msgctl

Synopsis: Perform various operations on a message queue.

Declaration: `function msgctl(msqid: cint; cmd: cint; buf: TMSQid_ds) : cint`

Visibility: `default`

Description: `msgctl` performs various operations on the message queue with id `ID`. Which operation is performed, depends on the `cmd` parameter, which can have one of the following values:

IPC_STAT In this case, the `msgctl` call fills the `TMSQid_ds` structure with information about the message queue.

IPC_SET in this case, the `msgctl` call sets the permissions of the queue as specified in the `ipc_perm` record inside `buf`.

IPC_RMID If this is specified, the message queue will be removed from the system.

`buf` contains the data that are needed by the call. It can be `Nil` in case the message queue should be removed.

The function returns `True` if success full, `False` otherwise.

Errors: On error, `False` is returned, and `IPCError` is set accordingly.

See also: `msgget` (953), `msgsnd` (954), `msgrcv` (954)

Listing: `./examples/ipcex/msgtool.pp`

program `msgtool`;

Uses `ipc,baseunix`;

Type

`PMyMsgBuf` = `^TMyMsgBuf`;
`TMyMsgBuf` = **record**
 `mtype` : `clong`;
 `mtext` : **string**[255];
end;

Procedure `DoError` (**Const** `Msg` : **string**);

begin

Writeln (`msg`, ' returned an error : ', `fpgeterrno`);


```

    halt(1);
end;

Procedure SendMessage (Id : Longint;
                        Var Buf : TMyMsgBuf;
                        MType : Longint;
                        Const MText : String);

begin
    Writeln ('Sending message. ');
    Buf.mtype:=mtype;
    Buf.Mtext:=mtext;
    If msgsnd(Id,PMsgBuf(@Buf),256,0)=-1 then
        DoError('msgsnd');
end;

Procedure ReadMessage (ID : Longint;
                        Var Buf : TMyMsgBuf;
                        MType : longint);

begin
    Writeln ('Reading message. ');
    Buf.MType:=MType;
    If msgrcv(ID,PMSGBuf(@Buf),256,mtype,0)<>-1 then
        Writeln ('Type : ',buf.mtype,' Text : ',buf.mtext)
    else
        DoError ('msgrcv');
end;

Procedure RemoveQueue ( ID : Longint);

begin
    If msgctl (id,IPC_RMID,Nil)<>-1 then
        Writeln ('Removed Queue with id ',Id);
end;

Procedure ChangeQueueMode (ID,mode : longint);

Var QueueDS : TMSQid_ds;

begin
    If msgctl (Id,IPC_STAT,@QueueDS)=-1 then
        DoError ('msgctl : stat');
    Writeln ('Old permissions : ',QueueDS.msg_perm.mode);
    QueueDS.msg_perm.mode:=Mode;
    if msgctl (ID,IPC_SET,@QueueDS)=0 then
        Writeln ('New permissions : ',QueueDS.msg_perm.mode)
    else
        DoError ('msgctl : IPC_SET');
end;

procedure usage;

begin
    Writeln ('Usage : msgtool s(end) <type> <text> (max 255 characters)');
    Writeln (' r(eceive) <type>');
    Writeln (' d(elete)');
    Writeln (' m(ode) <decimal mode>');

```

```

    halt(1);
end;

Function StrToInt (S : String): longint;

Var M : longint;
    C : Integer;

begin
    val (S,M,C);
    If C<>0 Then DoError ('StrToInt : '+S);
    StrToInt:=M;
end;

Var
    Key : TKey;
    ID : longint;
    Buf : TMyMsgBuf;

const ipckey = '.#0;

begin
    If Paramcount<1 then Usage;
    key :=Ftok(@ipckey[1],ord('M'));
    ID:=msgget(key,IPC_CREAT or 438);
    If ID<0 then DoError ('MsgGet');
    Case upCase(Paramstr(1)[1]) of
        'S' : If ParamCount<>3 then
            Usage
        else
            SendMessage (id,Buf,StrToInt(Paramstr(2)),paramstr(3));
        'R' : If ParamCount<>2 then
            Usage
        else
            ReadMessage (id,buf,strtoint(Paramstr(2)));
        'D' : If ParamCount<>1 then
            Usage
        else
            RemoveQueue (ID);
        'M' : If ParamCount<>2 then
            Usage
        else
            ChangeQueueMode (id,strtoint(paramstr(2)));
    else
        Usage
    end;
end.

```

58.4.3 msgget

Synopsis: Return message queue ID, possibly creating the queue.

Declaration: `function msgget (key: TKey; msgflg: cint) : cint`

Visibility: default

Description: `msgget` returns the ID of the message queue described by `key`. Depending on the flags in `msgflg`, a new queue is created.

`msgflg` can have one or more of the following values (combined by ORs):

IPC_CREAT The queue is created if it doesn't already exist.

IPC_EXCL If used in combination with **IPC_CREAT**, causes the call to fail if the queue already exists. It cannot be used by itself.

Optionally, the flags can be ORed with a permission mode, which is the same mode that can be used in the file system.

For an example, see `msgctl` (951).

Errors: On error, -1 is returned, and `IPCError` is set.

See also: `ftok` (950), `msgsnd` (954), `msgrcv` (954), `msgctl` (951)

58.4.4 `msgrcv`

Synopsis: Retrieve a message from the queue.

Declaration: `function msgrcv(msqid: cint; msgp: PMSGbuf; msgsz: size_t; msgtyp: clong; msgflg: cint) : cint`

Visibility: default

Description: `msgrcv` retrieves a message of type `msgtyp` from the message queue with ID `msqid`. `msgtyp` corresponds to the `mtype` field of the `TMSGbuf` record. The message is stored in the `MSGbuf` structure pointed to by `msgp`.

The `msgflg` parameter can be used to control the behaviour of the `msgrcv` call. It consists of an ORed combination of the following flags:

0 No special meaning.

IPC_NOWAIT if no messages are available, then the call returns immediately, with the `ENOMSG` error.

MSG_NOERROR If the message size is wrong (too large), no error is generated, instead the message is truncated. Normally, in such cases, the call returns an error (`E2BIG`)

The function returns `True` if the message was received correctly, `False` otherwise.

For an example, see `msgctl` (951).

Errors: In case of error, `False` is returned, and `IPCError` is set.

See also: `msgget` (953), `msgsnd` (954), `msgctl` (951)

58.4.5 `msgsnd`

Synopsis: Send a message to the message queue.

Declaration: `function msgsnd(msqid: cint; msgp: PMSGbuf; msgsz: size_t; msgflg: cint) : cint`

Visibility: default

Description: `msgsnd` sends a message to a message queue with ID `msqid`. `msgp` is a pointer to a message buffer, that should be based on the `TMsgBuf` type. `msgsz` is the size of the message (NOT of the message buffer record !)

The `msgflg` can have a combination of the following values (ORed together):

0 No special meaning. The message will be written to the queue. If the queue is full, then the process is blocked.

IPC_NOWAIT If the queue is full, then no message is written, and the call returns immediately.

The function returns `True` if the message was sent successfully, `False` otherwise.

For an example, see `msgctl` (951).

Errors: In case of error, the call returns `False`, and `IPCError` is set.

See also: `msgget` (953), `msgrcv` (954), `msgctl` (951)

58.4.6 `semctl`

Synopsis: Perform various control operations on a semaphore set.

Declaration: `function semctl(semid: cint; semnum: cint; cmd: cint; var arg: TSEMun) : cint`

Visibility: `default`

Description: `semctl` performs various operations on the semaphore `semnum` with semaphore set id `ID`.

The `arg` parameter supplies the data needed for each call. This is a variant record that should be filled differently, according to the command:

```
Type
TSEMun = record
  case longint of
    0 : ( val : longint );
    1 : ( buf : PSEMid_ds );
    2 : ( arr : PWord );
    3 : ( padbuf : PSeminfo );
    4 : ( padpad : pointer );
  end;
```

Which operation is performed, depends on the `cmd` parameter, which can have one of the following values:

IPC_STAT In this case, the `arg` record should have its `buf` field set to the address of a `TSEMid_ds` record. The `semctl` call fills this `TSEMid_ds` structure with information about the semaphore set.

IPC_SET In this case, the `arg` record should have its `buf` field set to the address of a `TSEMid_ds` record. The `semctl` call sets the permissions of the queue as specified in the `ipc_perm` record.

IPC_RMID If this is specified, the semaphore set is removed from the system.

GETALL In this case, the `arr` field of `arg` should point to a memory area where the values of the semaphores will be stored. The size of this memory area is `SizeOf(Word) * Number of semaphores in the set`. This call will then fill the memory array with all the values of the semaphores.

GETNCNT This will fill the `val` field of the `arg` union with the number of processes waiting for resources.

GETPID `semctl` returns the process ID of the process that performed the last `semop` (960) call.

GETVAL `semctl` returns the value of the semaphore with number `semnum`.

GETZCNT `semctl` returns the number of processes waiting for semaphores that reach value zero.

SETALL In this case, the `arr` field of `arg` should point to a memory area where the values of the semaphores will be retrieved from. The size of this memory area is `SizeOf(Word) * Number of semaphores in the set`. This call will then set the values of the semaphores from the memory array.

SETVAL This will set the value of semaphore `semnum` to the value in the `val` field of the `arg` parameter.

The function returns -1 on error.

Errors: The function returns -1 on error, and `IPCError` is set accordingly.

See also: `semget` (960), `semop` (960)

Listing: `./examples/ipcex/semtool.pp`

Program `semtool`;

{ Program to demonstrate the use of semaphores }

Uses `ipc,baseunix`;

Const `MaxSemValue = 5`;

Procedure `DoError (Const Msg : String)`;

var

`error: cint`;

begin

`error:=fpgeterrno`;

Writeln ('Error : ',msg,' Code : ',error);

Halt(1);

end;

Function `getsemval (ID,Member : longint) : longint`;

Var `S : TSEMun`;

begin

`GetSemVal:=SemCtl(id,member,SEM_GETVAL,S)`;

end;

Procedure `DispVal (ID,member : longint)`;

begin

writeln ('Value for member ',member,' is ',GetSemVal(ID,Member));

end;

Function `GetMemberCount (ID : Longint) : longint`;

Var `opts : TSEMun`;

`semds : TSEMid_ds`;

begin

`opts.buf:=@semds`;

If `semctl(Id,0,IPC_STAT,opts)<>-1` **then**

`GetMemberCount:=semds.sem_nsems`

else

`GetMemberCount:=-1`;

end;

Function OpenSem (Key : TKey) : Longint;

begin

OpenSem:=semget(Key,0,438);

If OpenSem=-1 **then**

DoError ('OpenSem');

end;

Function CreateSem (Key : TKey; Members : Longint) : Longint;

Var Count : Longint;

Semopts : TSemun;

begin

// the semmsl constant seems kernel specific

{ If members>semmsl then

DoError ('Sorry, maximum number of semaphores in set exceeded');

}

Writeln ('Trying to create a new semaphore set with ',members,' members.');

CreateSem:=semget(key,members,IPC_CREAT or IPC_Excl or 438);

If CreateSem=-1 **then**

DoError ('Semaphore set already exists.');

Semopts.val:=MaxSemValue; *{ Initial value of semaphores }*

For Count:=0 **to** Members-1 **do**

semctl(CreateSem,count,SEM_SETVAL,semopts);

end;

Procedure lockSem (ID,Member: Longint);

Var lock : TSEMbuf;

begin

With lock **do**

begin

sem_num:=0;

sem_op:=-1;

sem_flg:=IPC_NOWAIT;

end;

if (member<0) **or** (member>GetMemberCount(ID)-1) **then**

DoError ('semaphore member out of range');

if getsemval(ID,member)=0 **then**

DoError ('Semaphore resources exhausted (no lock)');

lock.sem_num:=member;

Writeln ('Attempting to lock member ',member, ' of semaphore ',ID);

if semop(Id,@lock,1)=-1 **then**

DoError ('Lock failed')

else

Writeln ('Semaphore resources decremented by one');

dispval(ID,Member);

end;

Procedure UnlockSem (ID,Member: Longint);

Var Unlock : TSEMbuf;

begin

```

With Unlock do
  begin
    sem_num:=0;
    sem_op:=1;
    sem_flg:=IPC_NOWAIT;
  end;
  if (member<0) or (member>GetMemberCount(ID)-1) then
    DoError ('semaphore member out of range');
  if getsemval(ID,member)=MaxSemValue then
    DoError ('Semaphore not locked');
  Unlock.sem_num:=member;
  Writeln ('Attempting to unlock member ',member, ' of semaphore ',ID);
  if semop(Id,@unlock,1)=-1 then
    DoError ('Unlock failed')
  else
    Writeln ('Semaphore resources incremented by one');
    dispval(ID,Member);
end;

```

```

Procedure RemoveSem (ID : longint);

```

```

var S : TSemun;

```

```

begin
  if semctl(Id,0,IPC_RMID,s)<>-1 then
    Writeln ('Semaphore removed')
  else
    DoError ('Couldn"t remove semaphore');
end;

```

```

Procedure ChangeMode (ID,Mode : longint);

```

```

Var rc : longint;
    opts : TSEMun;
    semds : TSEMid_ds;

```

```

begin
  opts.buf:=@semds;
  if not semctl (Id,0,IPC_STAT,opts)<>-1 then
    DoError ('Couldn"t stat semaphore');
  Writeln ('Old permissions were : ',semds.sem_perm.mode);
  semds.sem_perm.mode:=mode;
  if semctl(id,0,IPC_SET,opts)<>-1 then
    Writeln ('Set permissions to ',mode)
  else
    DoError ('Couldn"t set permissions');
end;

```

```

Procedure PrintSem (ID : longint);

```

```

Var l,cnt : longint;

```

```

begin
  cnt:=getmembercount(ID);
  Writeln ('Semaphore ',ID,' has ',cnt,' Members');
  For l:=0 to cnt-1 Do
    DispVal(id,l);

```

end;

Procedure USage;

begin

Writeln ('Usage : semtool c(reate) <count>');

Writeln (' l(ock) <member>');

Writeln (' u(nlock) <member>');

Writeln (' d(elete)');

Writeln (' m(ode) <mode>');

Writeln (' p(rint)');

halt(1);

end;

Function StrToInt (S : **String**): longint;

Var M : longint;

 C : Integer;

begin

val (S,M,C);

If C<>0 **Then** DoError ('StrToInt : '+S);

StrToInt:=M;

end;

Var Key : TKey;

 ID : Longint;

const ipckey='.#0;

begin

If ParamCount<1 **then** USage;

 key:=ftok(@ipckey[1],ORD('s'));

Case UpCase(Paramstr(1)[1]) **of**

 'C' : **begin**

if paramcount<>2 **then** usage;

 CreateSem (key,strtoint(paramstr(2)));

end;

 'L' : **begin**

if paramcount<>2 **then** usage;

 ID:=OpenSem (key);

 LockSem (ID,strtoint(paramstr(2)));

end;

 'U' : **begin**

if paramcount<>2 **then** usage;

 ID:=OpenSem (key);

 UnLockSem (ID,strtoint(paramstr(2)));

end;

 'M' : **begin**

if paramcount<>2 **then** usage;

 ID:=OpenSem (key);

 ChangeMode (ID,strtoint(paramstr(2)));

end;

 'D' : **Begin**

 ID:=OpenSem(Key);

 RemoveSem(Id);

end;

```

'P' : begin
    ID:=OpenSem(Key);
    PrintSem(Id);
    end;
else
    Usage
end;
end.

```

58.4.7 semget

Synopsis: Return the ID of a semaphore set, possibly creating the set.

Declaration: `function semget(key: TKey; nsems: cint; semflg: cint) : cint`

Visibility: default

Description: `msgget` returns the ID of the semaphore set described by `key`. Depending on the flags in `semflg`, a new queue is created.

`semflg` can have one or more of the following values (combined by ORs):

IPC_CREAT The queue is created if it doesn't already exist.

IPC_EXCL If used in combination with **IPC_CREAT**, causes the call to fail if the set already exists. It cannot be used by itself.

Optionally, the flags can be ORed with a permission mode, which is the same mode that can be used in the file system.

if a new set of semaphores is created, then there will be `nsems` semaphores in it.

Errors: On error, -1 is returned, and `IPCError` is set.

See also: `ftok` ([950](#)), `semop` ([960](#)), `semctl` ([955](#))

58.4.8 semop

Synopsis: Perform semaphore operation.

Declaration: `function semop(semid: cint; sops: PSEMbuf; nsops: cuint) : cint`

Visibility: default

Description: `semop` performs a set of operations on a message queue. `sops` points to an array of type `TSEMbuf`. The array should contain `nsops` elements.

The fields of the `TSEMbuf` ([966](#)) structure

```

TSEMbuf = record
    sem_num : word;
    sem_op  : integer;
    sem_flg : integer;

```

should be filled as follows:

sem_num The number of the semaphore in the set on which the operation must be performed.

sem_op The operation to be performed. The operation depends on the sign of `sem_op`: A positive number is simply added to the current value of the semaphore. If 0 (zero) is specified, then the process is suspended until the specified semaphore reaches zero. If a negative number is specified, it is subtracted from the current value of the semaphore. If the value would become negative then the process is suspended until the value becomes big enough, unless `IPC_NOWAIT` is specified in the `sem_flg`.

sem_flg Optional flags: if `IPC_NOWAIT` is specified, then the calling process will never be suspended.

The function returns `True` if the operations were successful, `False` otherwise.

Errors: In case of error, `False` is returned, and `IPCError` is set.

See also: `semget` (960), `semctl` (955)

58.4.9 semtimedop

Synopsis: Perform semaphore operation using timeout.

Declaration: `function semtimedop(semid: cint; sops: PSEMbuf; nsops: cuint; timeout: timespec) : cint`

Visibility: default

Description: `semop` performs a set of operations on a message queue, just as `semop` (960). `sops` points to an array of type `TSEMbuf`. The array should contain `nsops` elements. The `timeout` argument points to a time specification: if the operations cannot be performed within the specified time, the function will return with an error.

For more information on the actual operations, see `semop` (960).

See also: `semop` (960)

58.4.10 shmat

Synopsis: Attach a shared memory block.

Declaration: `function shmat(shmid: cint; shmaddr: pointer; shmflg: cint) : pointer`

Visibility: default

Description: `shmat` attaches a shared memory block with identified `shmid` to the current process. The function returns a pointer to the shared memory block.

If `shmaddr` is `Nil`, then the system chooses a free unmapped memory region, as high up in memory space as possible.

If `shmaddr` is non-nil, and `SHM_RND` is in `shmflg`, then the returned address is `shmaddr`, rounded down to `SHMLBA`. If `SHM_RND` is not specified, then `shmaddr` must be a page-aligned address.

The parameter `shmflg` can be used to control the behaviour of the `shmat` call. It consists of a ORed combination of the following constants:

SHM_RND The suggested address in `shmaddr` is rounded down to `SHMLBA`.

SHM_RDONLY the shared memory is attached for read access only. Otherwise the memory is attached for read-write. The process then needs read-write permissions to access the shared memory.

For an example, see `shmctl` (962).

Errors: If an error occurs, -1 is returned, and `IPCError` is set.

See also: `shmget` (964), `shmdt` (964), `shmctl` (962)

58.4.11 `shmctl`

Synopsis: Perform control operations on a shared memory block.

Declaration: `function shmctl(shmid: cint; cmd: cint; buf: PShmid_DS) : cint`

Visibility: default

Description: `shmctl` performs various operations on the shared memory block identified by identifier `shmid`.

The `buf` parameter points to a `TSHMid_ds` record. The `cmd` parameter is used to pass which operation is to be performed. It can have one of the following values :

IPC_STAT `shmctl` fills the `TSHMid_ds` record that `buf` points to with the available information about the shared memory block.

IPC_SET applies the values in the `ipc_perm` record that `buf` points to, to the shared memory block.

IPC_RMID the shared memory block is destroyed (after all processes to which the block is attached, have detached from it).

If successful, the function returns `True`, `False` otherwise.

Errors: If an error occurs, the function returns `False`, and `IPCError` is set.

See also: `shmget` (964), `shmat` (961), `shmdt` (964)

Listing: `./examples/ipcex/shmtool.pp`

```

Program shmtool;

uses ipc,strings,Baseunix;

Const SegSize = 100;

var key : Tkey;
    shmid,cntr : longint;
    segptr : pchar;

Procedure USage;

begin
  Writeln ('Usage : shmtool w(rite) text');
  writeln (' r(ead)');
  writeln (' d(elete)');
  writeln (' m(ode change) mode');
  halt(1);
end;

Procedure Writeshm (ID : Longint; ptr : pchar; S : string);

begin
  strcpy (ptr,S);

```

end;

Procedure Readshm(ID : longint; **ptr** : pchar);

begin

Writeln ('Read : ',**ptr**);

end;

Procedure removeshm (ID : Longint);

begin

 shmctl (ID,IPC_RMID,**Nil**);

writeln ('Shared memory marked for deletion');

end;

Procedure CHangeMode (ID : longint; mode : **String**);

Var m : word;

 code : integer;

 data : TSHMid_ds;

begin

val (mode,m,code);

if code<>0 **then**

 usage;

If shmctl (shmctl,IPC_STAT,@data)=-1 **then**

begin

writeln ('Error : shmctl : ',fpgeterrno);

halt(1);

end;

writeln ('Old permissions : ',data.shm_perm.mode);

 data.shm_perm.mode:=m;

If shmctl (shmctl,IPC_SET,@data)=-1 **then**

begin

writeln ('Error : shmctl : ',fpgeterrno);

halt(1);

end;

writeln ('New permissions : ',data.shm_perm.mode);

end;

const ftokpath = '#0';

begin

if paramcount<1 **then** usage;

 key := ftok (pchar(@ftokpath[1]),ord('S'));

 shmctl := shmget(key,segsz,IPC_CREAT or IPC_EXCL or 438);

If shmctl=-1 **then**

begin

Writeln ('Shared memory exists. Opening as client');

 shmctl := shmget(key,segsz,0);

If shmctl = -1 **then**

begin

Writeln ('shmget : Error !',fpgeterrno);

halt(1);

end

end

else

Writeln ('Creating new shared memory segment.');

```

segptr:=shmat(shmid,nil,0);
if longint(segptr)==-1 then
  begin
    Writeln ('Shmat : error !',fpgeterrno);
    halt(1);
  end;
case upcase(paramstr(1)[1]) of
  'W' : writeshm (shmid,segptr,paramstr(2));
  'R' : readshm (shmid,segptr);
  'D' : removeshm(shmid);
  'M' : changemode (shmid,paramstr(2));
else
  begin
    writeln (paramstr(1));
    usage;
  end;
end;
end.

```

58.4.12 shmdt

Synopsis: Detach shared memory block.

Declaration: `function shmdt (shmaddr: pointer) : cint`

Visibility: default

Description: `shmdt` detaches the shared memory at address `shmaddr`. This shared memory block is unavailable to the current process, until it is attached again by a call to `shmat` (961).

The function returns `True` if the memory block was detached successfully, `False` otherwise.

Errors: On error, `False` is returned, and `IPCError` is set.

See also: `shmget` (964), `shmat` (961), `shmctl` (962)

58.4.13 shmget

Synopsis: Return the ID of a shared memory block, possibly creating it.

Declaration: `function shmget (key: TKey; size: size_t; flag: cint) : cint`

Visibility: default

Description: `shmget` returns the ID of a shared memory block, described by `key`. Depending on the flags in `flag`, a new memory block is created.

`flag` can have one or more of the following values (combined by ORs):

IPC_CREAT The queue is created if it doesn't already exist.

IPC_EXCL If used in combination with `IPC_CREAT`, causes the call to fail if the queue already exists. It cannot be used by itself.

Optionally, the flags can be ORed with a permission mode, which is the same mode that can be used in the file system.

if a new memory block is created, then it will have size `Size` bytes in it.

Errors: On error, -1 is returned, and `IPCError` is set.

58.5 TIPC_Perm

```
TIPC_Perm = record
  key : TKey;
  uid : kernel_uid_t;
  gid : kernel_gid_t
  ;
  cuid : kernel_uid_t;
  cgid : kernel_gid_t;
  mode : kernel_mode_t
  ;
  __pad1 : Array[1..4-sizeof(mode_t)] of Byte;
  seq : cushort;
  __pad2 : cushort;
  __unused1 : culong;
  __unused2 : culong;
end
```

TIPC_Perm is used in all IPC systems to specify the permissions. It should never be used directly.

58.6 TMSG

```
TMSG = record
  msg_next : PMSG;
  msg_type : LongInt;
  msg_spot
  : PChar;
  msg_stime : LongInt;
  msg_ts : Integer;
end
```

Record used in the handling of message queues. Do not use directly.

58.7 TMSGbuf

```
TMSGbuf = record
  mtype : clong;
  mtext : Array[0..0] of Char;
end
```

The TMSGbuf record is a record containing the data of a record. you should never use this record directly, instead you should make your own record that follows the structure of the TMSGbuf record, but that has a size that is big enough to accommodate your messages. The mtype field should always be present, and should always be filled.

58.8 TMSGinfo

```
TMSGinfo = record
```

```

msgpool : cint;
msgmap : cint;
msgmax : cint
;
msgmnb : cint;
msgmni : cint;
msgssz : cint;
msgtql : cint
;
msgseg : cushort;
end

```

Internal message system record. Do not use directly.

58.9 TMSQid_ds

```

TMSQid_ds = record
  msg_perm : TIPC_Perm;
  msg_stime : time_t;
  msg_rtime : time_t;
  msg_ctime : time_t;
  msg_cbytes : QWord;
  msg_qnum : QWord;
  msg_qbytes : QWord;
  msg_lspid : ipc_pid_t
;
  msg_lrpid : ipc_pid_t;
  pad1 : QWord;
  pad2 : QWord;
end

```

This record should never be used directly, it is an internal kernel record. It's fields may change at any time.

58.10 TSEMbuf

```

TSEMbuf = record
  sem_num : cushort;
  sem_op : cshort;
  sem_flg
  : cshort;
end

```

The TSEMbuf record is used in the semop (960) call, and is used to specify which operations you want to do.

58.11 TSEMid_ds

```

TSEMid_ds = record

```

```

sem_perm : TIPC_Perm;
sem_otime : time_t;
unused1 : culong;
sem_ctime : time_t;
unused2 : culong;
sem_nsems
: culong;
unused3 : culong;
unused4 : culong;
end

```

Structure returned by the `semctl` (955) call, contains all data of a semaphore.

58.12 TSEMinfo

```

TSEMinfo = record
  semmap : cint;
  semmni : cint;
  semmns : cint
  ;
  semmnu : cint;
  semmsl : cint;
  semopm : cint;
  semume : cint
  ;
  semusz : cint;
  semvmx : cint;
  semaem : cint;
end

```

Internal semaphore system record. Do not use.

58.13 TShmid_ds

```

TShmid_ds = record
  shm_perm : TIPC_Perm;
  shm_segsz : size_t;
  shm_atime : time_t;
  shm_dtime : time_t;
  shm_ctime : time_t;
  shm_cpid : pid_t;
  shm_lpid : pid_t;
  shm_nattch : culong;
  __unused4 : culong;
  __unused5 : culong;
end

```

Record used in the `shmctl` (962) call to set or retrieve settings for shared memory.

58.14 TSHMinfo

```
TSHMinfo = record
  shmmax : cint;
  shmmmin : cint;
  shmmni : cint
  ;
  shmseg : cint;
  shmall : cint;
end
```

Record used by the shared memory system, Do not use directly.

58.15 TSHM_info

```
TSHM_info = record
  used_ids : cint;
  shm_tot : culong;
  shm_rss
  : culong;
  shm_swp : culong;
  swap_attempts : culong;
  swap_successes
  : culong;
end
```

Chapter 59

Reference for unit 'keyboard'

59.1 Used units

Table 59.1: Used units by unit 'keyboard'

Name	Page
System	1335

59.2 Overview

The **Keyboard** unit implements a keyboard access layer which is system independent. It can be used to poll the keyboard state and wait for certain events. Waiting for a keyboard event can be done with the `GetKeyEvent` ([983](#)) function, which will return a driver-dependent key event. This key event can be translated to a interpretable event by the `TranslateKeyEvent` ([991](#)) function. The result of this function can be used in the other event examining functions.

A custom keyboard driver can be installed using the `SetKeyboardDriver` ([990](#)) function. The current keyboard driver can be retrieved using the `GetKeyboardDriver` ([982](#)) function. The last section of this chapter demonstrates how to make a keyboard driver.

59.3 Unix specific notes.

On Unix, applications run on a "terminal", and the application writes to the screen and reads from the keyboard by communicating with the terminal. Unix keyboard handling is mostly backward compatible with the DEC VT100 and VT220 terminals from tens of years ago. The VT100 and VT220 had very different keyboards than today's PC's and this is where the problems start. To make it worse the protocol of both terminals has not been very well designed.

Because of this, the keyboard unit on Unix operating systems does a best effort to provide keyboard functionality. An implementation with full keyboard facilities like on other operating systems is not possible.

The exception is the Linux kernel. The terminal emulation of the Linux kernel is from a PC keyboard viewpoint hopeless as well, but unlike other terminal emulators it is configurable. On the Linux console, the Free Pascal keyboard unit tries to implement full functionality.

Users of applications using the keyboard unit should expect the following:

- Full functionality on the Linux console. It must be the bare console, SSH into another machine will kill the full functionality.
- Limited functionality otherwise.

Notes about Linux full functionality:

- The keyboard is reprogrammed. If the keyboard is for whatever reason not restored in its original state, please load your keymap to reinitialize it.
- Alt+function keys generate keycodes for those keys. To switch virtual consoles, use ctrl+alt+function key.
- Unlike what you're used to with other Unix software, escape works as you intuitively expect, it generates the keycode for an escape key **without a delay**.

The limited functionality does include these quirks:

- Escape must be pressed two times before it has effect.
- On the Linux console, when the users runs the program by logging into another machine:
 - Shift+F1 and Shift+F12 will generate keycodes for F11 and F12.
 - Shift+arrow keys, shift+ins, shift+del, shift+home, shift+end do not work. The same is true about the control and alt combinations.
 - Alt+function keys will switch virtual consoles instead of generating the right key sequences.
 - Ctrl+function keys will generate the keycodes for the function keys without ctrl
- In Xterm:
 - Shift+insert pastes the x clipboard, no keycode will be generated.
- In Konsole:
 - Shift+insert pastes the x clipboard, no keycode will be generated.
 - Shift+arrow keys doesn't work, nor does ctrl+arrow keys

If you have a non-standard terminal, some keys may not work at all. When in limited functionality mode, the user can work around using an escape prefix:

- Esc+1 = F1, Esc+2 = F2.
- Esc before another key is equal to alt+key.

In such cases, if the terminal does output an escape sequence for those keys, please submit a bug report so we can add them.

59.4 Writing a keyboard driver.

Writing a keyboard driver means that hooks must be created for most of the keyboard unit functions. The `TKeyboardDriver` record contains a field for each of the possible hooks:

```
TKeyboardDriver = Record
  InitDriver : Procedure;
  DoneDriver : Procedure;
  GetKeyEvent : Function : TKeyEvent;
  PollKeyEvent : Function : TKeyEvent;
  GetShiftState : Function : Byte;
  TranslateKeyEvent : Function (KeyEvent: TKeyEvent): TKeyEvent;
  TranslateKeyEventUniCode: Function (KeyEvent: TKeyEvent): TKeyEvent;
end;
```

The meaning of these hooks is explained below:

InitDriver Called to initialize and enable the driver. Guaranteed to be called only once. This should initialize all needed things for the driver.

DoneDriver Called to disable and clean up the driver. Guaranteed to be called after a call to `initDriver`. This should clean up all things initialized by `InitDriver`.

GetKeyEvent Called by `GetKeyEvent` (983). Must wait for and return the next key event. It should NOT store keys.

PollKeyEvent Called by `PollKeyEvent` (988). It must return the next key event if there is one. Should not store keys.

GetShiftState Called by `PollShiftStateEvent` (988). Must return the current shift state.

TranslateKeyEvent Should translate a raw key event to a correct key event, i.e. should fill in the shiftstate and convert function key scancodes to function key keycodes. If the `TranslateKeyEvent` is not filled in, a default translation function will be called which converts the known scancodes from the tables in the previous section to a correct keyevent.

TranslateKeyEventUniCode Should translate a key event to a UNICode key representation.

Strictly speaking, only the `GetKeyEvent` and `PollKeyEvent` hooks must be implemented for the driver to function correctly.

The example unit demonstrates how a keyboard driver can be installed. It takes the installed driver, and hooks into the `GetKeyEvent` function to register and log the key events in a file. This driver can work on top of any other driver, as long as it is inserted in the `uses` clause *after* the real driver unit, and the real driver unit should set the driver record in its initialization section.

Note that with a simple extension of this unit could be used to make a driver that is capable of recording and storing a set of keyboard strokes, and replaying them at a later time, so a 'keyboard macro' capable driver. This driver could sit on top of any other driver.

Listing: `./examples/kbdex/logkeys.pp`

```
unit logkeys;
```

```
interface
```

```
Procedure StartKeyLogging;
```

```
Procedure StopKeyLogging;
```

Function IsKeyLogging : Boolean;

Procedure SetKeyLogFileName(FileName : **String**);

implementation

uses sysutils,keyboard;

var

 NewKeyBoardDriver,
 OldKeyBoardDriver : TKeyboardDriver;
 Active,Logging : Boolean;
 LogFileName : **String**;
 KeyLog : Text;

Function TimeStamp : **String**;

begin

 TimeStamp:=**FormatDateTime**('hh:nn:ss',**Time**());

end;

Procedure StartKeyLogging;

begin

 Logging:=True;
 WriteLn(KeyLog,'Start logging keystrokes at: ',TimeStamp);
end;

Procedure StopKeyLogging;

begin

WriteLn(KeyLog,'Stop logging keystrokes at: ',TimeStamp);
 Logging:=False;
end;

Function IsKeyLogging : Boolean;

begin

 IsKeyLogging:=Logging;
end;

Function LogGetKeyEvent : TKeyEvent;

Var

 K : TKeyEvent;

begin

 K:=OldkeyboardDriver.GetKeyEvent();
 If Logging **then**
 begin
 Write(KeyLog,TimeStamp,': Key event: ');
 WriteLn(KeyLog,KeyEventToString(TranslateKeyEvent(K)));
 end;
 LogGetKeyEvent:=K;
end;

Procedure LogInitKeyBoard;

```

begin
  OldKeyBoardDriver.InitDriver();
  Assign(KeyLog,logFileName);
  Rewrite(KeyLog);
  Active:=True;
  StartKeyLogging;
end;

Procedure LogDoneKeyBoard;

begin
  StopKeyLogging;
  Close(KeyLog);
  Active:=False;
  OldKeyBoardDriver.DoneDriver();
end;

Procedure SetKeyLogFileName(FileName : String);

begin
  If Not Active then
    LogFileName:=FileName;
  end;

Initialization
  GetKeyBoardDriver(OldKeyBoardDriver);
  NewKeyBoardDriver:=OldKeyBoardDriver;
  NewKeyBoardDriver.GetKeyEvent:=@LogGetKeyEvent;
  NewKeyBoardDriver.InitDriver:=@LogInitKeyboard;
  NewKeyBoardDriver.DoneDriver:=@LogDoneKeyboard;
  LogFileName:='keyboard.log';
  Logging:=False;
  SetKeyboardDriver(NewKeyBoardDriver);
end.

```

Listing: ./examples/kbdex/ex9.pp

```

program example9;

{ This program demonstrates the logkeys unit }

uses keyboard,logkeys;

Var
  K : TKeyEvent;

begin
  InitKeyBoard;
  Writeln('Press keys, press "q" to end, "s" toggles logging. ');
  Repeat
    K:=GetKeyEvent;
    K:=TranslateKeyEvent(K);
    Writeln('Got key : ',KeyEventToString(K));
    if GetKeyEventChar(K)='s' then
      if IsKeyLogging then
        StopKeyLogging
      else
        StartKeyLogging;
    Until (GetKeyEventChar(K)='q');

```

```
DoneKeyBoard;  
end.
```

59.5 Keyboard scan codes.

Special physical keys are encoded with the DOS scan codes for these keys in the second byte of the TKeyEvent (981) type. A complete list of scan codes can be found in the below table. This is the list of keys that is used by the default key event translation mechanism. When writing a keyboard driver, either these constants should be returned by the various key event functions, or the TranslateKeyEvent hook should be implemented by the driver.

Table 59.2: Key Scancodes

Code	Key	Code	Key	Code	Key
00	NoKey	3D	F3	70	ALT-F9
01	ALT-Esc	3E	F4	71	ALT-F10
02	ALT-Space	3F	F5	72	CTRL-PrtSc
04	CTRL-Ins	40	F6	73	CTRL-Left
05	SHIFT-Ins	41	F7	74	CTRL-Right
06	CTRL-Del	42	F8	75	CTRL-end
07	SHIFT-Del	43	F9	76	CTRL-PgDn
08	ALT-Back	44	F10	77	CTRL-Home
09	ALT-SHIFT-Back	47	Home	78	ALT-1
0F	SHIFT-Tab	48	Up	79	ALT-2
10	ALT-Q	49	PgUp	7A	ALT-3
11	ALT-W	4B	Left	7B	ALT-4
12	ALT-E	4C	Center	7C	ALT-5
13	ALT-R	4D	Right	7D	ALT-6
14	ALT-T	4E	ALT-GrayPlus	7E	ALT-7
15	ALT-Y	4F	end	7F	ALT-8
16	ALT-U	50	Down	80	ALT-9
17	ALT-I	51	PgDn	81	ALT-0
18	ALT-O	52	Ins	82	ALT-Minus
19	ALT-P	53	Del	83	ALT-Equal
1A	ALT-LftBrack	54	SHIFT-F1	84	CTRL-PgUp
1B	ALT-RgtBrack	55	SHIFT-F2	85	F11
1E	ALT-A	56	SHIFT-F3	86	F12
1F	ALT-S	57	SHIFT-F4	87	SHIFT-F11
20	ALT-D	58	SHIFT-F5	88	SHIFT-F12
21	ALT-F	59	SHIFT-F6	89	CTRL-F11
22	ALT-G	5A	SHIFT-F7	8A	CTRL-F12
23	ALT-H	5B	SHIFT-F8	8B	ALT-F11
24	ALT-J	5C	SHIFT-F9	8C	ALT-F12
25	ALT-K	5D	SHIFT-F10	8D	CTRL-Up
26	ALT-L	5E	CTRL-F1	8E	CTRL-Minus
27	ALT-SemiCol	5F	CTRL-F2	8F	CTRL-Center
28	ALT-Quote	60	CTRL-F3	90	CTRL-GreyPlus
29	ALT-OpQuote	61	CTRL-F4	91	CTRL-Down
2B	ALT-BkSlash	62	CTRL-F5	94	CTRL-Tab
2C	ALT-Z	63	CTRL-F6	97	ALT-Home
2D	ALT-X	64	CTRL-F7	98	ALT-Up
2E	ALT-C	65	CTRL-F8	99	ALT-PgUp
2F	ALT-V	66	CTRL-F9	9B	ALT-Left
30	ALT-B	67	CTRL-F10	9D	ALT-Right
31	ALT-N	68	ALT-F1	9F	ALT-end
32	ALT-M	69	ALT-F2	A0	ALT-Down
33	ALT-Comma	6A	ALT-F3	A1	ALT-PgDn
34	ALT-Period	6B	ALT-F4	A2	ALT-Ins
35	ALT-Slash	6C	ALT-F5	A3	ALT-Del
37	ALT-GreyAst	6D	ALT-F6	A5	ALT-Tab
3B	F1	6E	ALT-F7		
3C	F2	6F	ALT-F8		

A list of scan codes for special keys and combinations with the SHIFT, ALT and CTRL keys can be found in the following table: They are for quick reference only.

Table 59.3: Special keys scan codes

Key	Code	SHIFT-Key	CTRL-Key	Alt-Key
NoKey	00			
F1	3B	54	5E	68
F2	3C	55	5F	69
F3	3D	56	60	6A
F4	3E	57	61	6B
F5	3F	58	62	6C
F6	40	59	63	6D
F7	41	5A	64	6E
F8	42	5B	65	6F
F9	43	5C	66	70
F10	44	5D	67	71
F11	85	87	89	8B
F12	86	88	8A	8C
Home	47		77	97
Up	48		8D	98
PgUp	49		84	99
Left	4B		73	9B
Center	4C		8F	
Right	4D		74	9D
end	4F		75	9F
Down	50		91	A0
PgDn	51		76	A1
Ins	52	05	04	A2
Del	53	07	06	A3
Tab	8	0F	94	A5
GreyPlus			90	4E

59.6 Constants, types and variables

59.6.1 Constants

`AltPrefix : Byte = 0`

Keycode for alternate prefix key for Alt key. Unix Only.

`CtrlPrefix : Byte = 0`

Keycode for alternate prefix key for Ctrl key. Unix only.

`errKbdBase = 1010`

Base of keyboard routine error reporting constants.

`errKbdInitError = errKbdBase + 0`

Failed to initialize keyboard driver.

```
errKbdNotImplemented = errKbdBase + 1
```

Keyboard driver not implemented.

```
kbAlt = 8
```

Alt key modifier.

```
kbASCII = $00
```

ASCII code key event.

```
kbCtrl = 4
```

Control key modifier.

```
kbdApps = $FF17
```

Application key (popup-menu) pressed.

```
kbdDelete = $FF2A
```

Delete key pressed.

```
kbdDown = $FF27
```

Arrow down key pressed.

```
kbdEnd = $FF26
```

End key pressed.

```
kbdF1 = $FF01
```

F1 function key pressed.

```
kbdF10 = $FF0A
```

F10 function key pressed.

```
kbdF11 = $FF0B
```

F12 function key pressed.

```
kbdF12 = $FF0C
```

F12 function key pressed.

```
kbdF13 = $FF0D
```

F13 function key pressed.

kbdF14 = \$FF0E

F14 function key pressed.

kbdF15 = \$FF0F

F15 function key pressed.

kbdF16 = \$FF10

F16 function key pressed.

kbdF17 = \$FF11

F17 function key pressed.

kbdF18 = \$FF12

F18 function key pressed.

kbdF19 = \$FF13

F19 function key pressed.

kbdF2 = \$FF02

F2 function key pressed.

kbdF20 = \$FF14

F20 function key pressed.

kbdF3 = \$FF03

F3 function key pressed.

kbdF4 = \$FF04

F4 function key pressed.

kbdF5 = \$FF05

F5 function key pressed.

kbdF6 = \$FF06

F6 function key pressed.

kbdF7 = \$FF07

F7 function key pressed.

`kbdF8 = $FF08`

F8 function key pressed.

`kbdF9 = $FF09`

F9 function key pressed.

`kbdHome = $FF20`

Home key pressed.

`kbdInsert = $FF29`

Insert key pressed.

`kbdLeft = $FF23`

Arrow left key pressed.

`kbdLWin = $FF15`

Left windows key pressed.

`kbdMiddle = $FF24`

Middle key pad key pressed (numerical 5).

`kbdPgDn = $FF28`

Page down key pressed.

`kbdPgUp = $FF22`

Page Up key pressed.

`kbdRight = $FF25`

Arrow right key pressed.

`kbdRWin = $FF16`

Right windows key pressed.

`kbdUp = $FF21`

Arrow up key pressed.

`kbFnKey = $02`

function key pressed.

```
kbLeftShift = 1
```

Left shift key modifier.

```
kbPhys = $03
```

Physical key code event.

```
kbReleased = $04
```

Key release event (not implemented in FPC).

```
kbRightShift = 2
```

Right shift key modifier.

```
kbShift = kbLeftShift or kbRightShift
```

Shift key modifier.

```
kbUnicode = $01
```

Unicode code key event.

```
SAnd : string = 'AND'
```

This constant is used as the 'And' word in key descriptions. This constant is used by the key event description routines. It can be changed to localize the key descriptions when needed.

```
ShiftPrefix : Byte = 0
```

Keycode for alternate prefix key for Shift key. Unix Only.

```
SKeyPad : Array[0..($FF2F-kbdHome)] of string = ('Home', 'Up', 'PgUp',  
    , 'Left', 'Middle', 'Right', 'End', 'Down', 'PgDn', 'Insert', 'Delete',  
    , '', '', '', '', '')
```

This constant describes all keypad keys. This constant is used by the key event description routines. It can be changed to localize the key descriptions when needed.

```
SLeftRight : Array[1..2] of string = ('LEFT', 'RIGHT')
```

This constant contains strings to describe left and right keys. This constant is used by the key event description routines. It can be changed to localize the key descriptions when needed.

```
SScanCode : string = 'Key with scancode '
```

This constant contains a string to denote a scancode key event. This constant is used by the key event description routines. It can be changed to localize the key descriptions when needed.

```
SShift : Array[1..3] of string = ('SHIFT', 'CTRL', 'ALT')
```

This constant describes the various modifier keys. This constant is used by the key event description routines. It can be changed to localize the key descriptions when needed.

```
SUnicodeChar : string = 'Unicode character '
```

This constant contains a string to denote a Unicode key event. This constant is used by the key event description routines. It can be changed to localize the key descriptions when needed.

```
SUnknownFunctionKey : string = 'Unknown function key : '
```

This constant contains a string to denote that an unknown function key was found. This constant is used by the key event description routines. It can be changed to localize the key descriptions when needed.

59.6.2 Types

```
PTreeElement = ^TTreeElement
```

Pointer to TTreeElement (993) record.

```
TKeyEvent = Cardinal
```

The TKeyEvent type is the base type for all keyboard events.

The key stroke is encoded in the 4 bytes of the TKeyEvent type. The various fields of the key stroke encoding can be obtained by typecasting the TKeyEvent type to the TKeyRecord (992) type.

```
Tprocedure = procedure
```

Procedure prototype.

59.7 Procedures and functions

59.7.1 DoneKeyboard

Synopsis: Deactivate keyboard driver.

Declaration: `procedure DoneKeyboard`

Visibility: default

Description: DoneKeyboard de-initializes the keyboard interface if the keyboard driver is active. If the keyboard driver is not active, the function does nothing.

This will cause the keyboard driver to clear up any allocated memory, or restores the console or terminal the program was running in to its initial state before the call to InitKeyBoard (986). This function should be called on program exit. Failing to do so may leave the terminal or console window in an unusable state. Its exact action depends on the platform on which the program is running.

On Unix the default keyboard driver restores the line ending of system.output to #10.

For an example, see most other functions.

Errors: None.

See also: InitKeyBoard (986)

59.7.2 FunctionKeyName

Synopsis: Return string representation of a function key code.

Declaration: `function FunctionKeyName(KeyCode: Word) : string`

Visibility: default

Description: `FunctionKeyName` returns a string representation of the function key with code `KeyCode`. This can be an actual function key, or one of the cursor movement keys.

Errors: In case `KeyCode` does not contain a function code, the `SUnknownFunctionKey` string is returned, appended with the `KeyCode`.

See also: `ShiftStateToString` (990), `KeyEventToString` (987)

Listing: `./examples/kbdex/ex8.pp`

Program Example8;

{ Program to demonstrate the FunctionKeyName function. }

Uses keyboard;

Var

 K : TKeyEvent;

begin

 InitKeyboard;

 WriteLn('Press function keys, press "q" to end.');

Repeat

 K:=GetKeyEvent;

 K:=TranslateKeyEvent(K);

If IsFunctionKey(k) **then**

begin

 Write('Got function key : ');

 WriteLn(FunctionKeyName(TkeyRecord(K).KeyCode));

end;

Until (GetKeyEventChar(K)='q');

 DoneKeyboard;

end.

59.7.3 GetKeyboardDriver

Synopsis: Return the current keyboard driver record.

Declaration: `procedure GetKeyboardDriver(var Driver: TKeyboardDriver)`

Visibility: default

Description: `GetKeyboardDriver` returns in `Driver` the currently active keyboard driver. This function can be used to enhance an existing keyboarddriver.

For more information on getting and setting the keyboard driver `kbd driver` (971).

Errors: None.

See also: `SetKeyboardDriver` (990)

59.7.4 GetKeyEvent

Synopsis: Get the next raw key event, wait if needed.

Declaration: `function GetKeyEvent : TKeyEvent`

Visibility: default

Description: `GetKeyEvent` returns the last keyevent if it is available, or waits for one if none is available. A non-blocking version is available in `PollKeyEvent` (988).

The returned key is encoded as a `TKeyEvent` type variable, and is normally the physical key scan code, (the scan code is driver dependent) which can be translated with one of the translation functions `TranslateKeyEvent` (991) or `TranslateKeyEventUnicode` (991). See the types section for a description of how the key is described.

Errors: If no key became available (e.g. when the driver does not support it), 0 is returned.

See also: `PutKeyEvent` (989), `PollKeyEvent` (988), `TranslateKeyEvent` (991), `TranslateKeyEventUnicode` (991)

Listing: `./examples/kbdex/ex1.pp`

```

program example1;

{ This program demonstrates the GetKeyEvent function }

uses keyboard;

Var
  K : TKeyEvent;

begin
  InitKeyboard;
  Writeln('Press keys, press "q" to end. ');
  Repeat
    K:=GetKeyEvent;
    K:=TranslateKeyEvent(K);
    Write('Got key event with ');
    Case GetKeyEventFlags(K) of
      kbASCII : Writeln('ASCII key');
      kbUnicode : Writeln('Unicode key');
      kbFnKey : Writeln('Function key');
      kbPhys : Writeln('Physical key');
      kbReleased : Writeln('Released key event');
    end;
    Writeln('Got key : ',KeyEventToString(K));
  Until (GetKeyEventChar(K)='q');
  DoneKeyboard;
end.

```

59.7.5 GetKeyEventChar

Synopsis: Get the character key part of a key event.

Declaration: `function GetKeyEventChar (KeyEvent : TKeyEvent) : Char`

Visibility: default

Description: `GetKeyEventChar` returns the charcode part of the given `KeyEvent`, if it contains a translated character key keycode. The charcode is simply the ascii code of the character key that was pressed.

It returns the null character if the key was not a character key, but e.g. a function key.

For an example, see `GetKeyEvent` (983)

Errors: None.

See also: `GetKeyEventUnicode` (986), `GetKeyEventShiftState` (985), `GetKeyEventFlags` (985), `GetKeyEventCode` (984), `GetKeyEvent` (983)

59.7.6 `GetKeyEventCode`

Synopsis: Translate function key part of a key event code.

Declaration: `function GetKeyEventCode (KeyEvent: TKeyEvent) : Word`

Visibility: default

Description: `GetKeyEventCode` returns the translated function keycode part of the given `KeyEvent`, if it contains a translated function key.

If the key pressed was not a function key, the null character is returned.

Errors: None.

See also: `GetKeyEventUnicode` (986), `GetKeyEventShiftState` (985), `GetKeyEventFlags` (985), `GetKeyEventChar` (983), `GetKeyEvent` (983)

Listing: `./examples/kbdex/ex2.pp`

Program Example2;

{ Program to demonstrate the GetKeyEventCode function. }

Uses keyboard;

Var

 K : TKeyEvent;

begin

 InitKeyBoard;

Writeln('Press function keys, or press "q" to end.');

Repeat

 K:=GetKeyEvent;

 K:=TranslateKeyEvent(K);

If (GetKeyEventFlags(K)<>KbfnKey) **then**

Writeln('Not a function key')

else

begin

Write('Got key (',GetKeyEventCode(K));

Writeln(') : ',KeyEventToString(K));

end;

Until (GetKeyEventChar(K)='q');

 DoneKeyboard;

end.

59.7.7 GetKeyEventFlags

Synopsis: Extract the flags from a key event.

Declaration: `function GetKeyEventFlags (KeyEvent: TKeyEvent) : Byte`

Visibility: default

Description: `GetKeyEventFlags` returns the flags part of the given `KeyEvent`.

For an example, see `GetKeyEvent` (983)

Errors: None.

See also: `GetKeyEventUnicode` (986), `GetKeyEventShiftState` (985), `GetKeyEventCode` (984), `GetKeyEventChar` (983), `GetKeyEvent` (983)

59.7.8 GetKeyEventShiftState

Synopsis: Return the current state of the shift keys.

Declaration: `function GetKeyEventShiftState (KeyEvent: TKeyEvent) : Byte`

Visibility: default

Description: `GetKeyEventShiftState` returns the shift-state values of the given `KeyEvent`. This can be used to detect which of the modifier keys `Shift`, `Alt` or `Ctrl` were pressed. If none were pressed, zero is returned.

Note that this function does not always return expected results; In a UNIX X-Term, the modifier keys do not always work.

Errors: None.

See also: `GetKeyEventUnicode` (986), `GetKeyEventFlags` (985), `GetKeyEventCode` (984), `GetKeyEventChar` (983), `GetKeyEvent` (983)

Listing: `./examples/kbdex/ex3.pp`

Program Example3;

{ Program to demonstrate the GetKeyEventShiftState function. }

Uses keyboard;

Var

 K : TKeyEvent;
 S : Byte;

begin

 InitKeyBoard;
 Write('Press keys combined with CTRL/SHIFT/ALT');
 Writeln(', or press "q" to end.');

Repeat
 K:=GetKeyEvent;
 K:=TranslateKeyEvent(K);
 S:=GetKeyEventShiftState(K);
 If (S=0) **then**
 Writeln('No special keys pressed')
 else
 begin

```

    Writeln('Detected special keys : ', ShiftStateToString(K, False));
    Writeln('Got key : ', KeyEventToString(K));
    end;
    Until (GetKeyEventChar(K)='q');
    DoneKeyboard;
end.

```

59.7.9 GetKeyEventUnicode

Synopsis: Return the Unicode key event.

Declaration: `function GetKeyEventUnicode(KeyEvent: TKeyEvent) : Word`

Visibility: default

Description: `GetKeyEventUnicode` returns the Unicode part of the given `KeyEvent` if it contains a translated Unicode character.

Errors: None.

See also: `GetKeyEventShiftState` (985), `GetKeyEventFlags` (985), `GetKeyEventCode` (984), `GetKeyEventChar` (983), `GetKeyEvent` (983)

59.7.10 InitKeyboard

Synopsis: Initialize the keyboard driver.

Declaration: `procedure InitKeyboard`

Visibility: default

Description: `InitKeyboard` initializes the keyboard driver. If the driver is already active, it does nothing. When the driver is initialized, it will do everything necessary to ensure the functioning of the keyboard, including allocating memory, initializing the terminal etc.

This function should be called once, before using any of the keyboard functions. When it is called, the `DoneKeyboard` (981) function should also be called before exiting the program or changing the keyboard driver with `SetKeyboardDriver` (990).

On Unix, the default keyboard driver sets terminal in raw mode. In raw mode the line feed behaves as an actual linefeed, i.e. the cursor is moved down one line. while the x coordinate does not change. To compensate, the default keyboard sets driver line ending of `system.output` to #13#10.

For an example, see most other functions.

Errors: None.

See also: `DoneKeyboard` (981), `SetKeyboardDriver` (990)

59.7.11 IsFunctionKey

Synopsis: Check whether a given event is a function key event.

Declaration: `function IsFunctionKey(KeyEvent: TKeyEvent) : Boolean`

Visibility: default

Description: `IsFunctionKey` returns `True` if the given key event in `KeyEvent` was a function key or not.

Errors: None.

See also: [GetKeyEvent \(983\)](#)

Listing: ./examples/kbdex/ex7.pp

```

program example1;

{ This program demonstrates the GetKeyEvent function }

uses keyboard;

Var
  K : TKeyEvent;

begin
  InitKeyBoard;
  Writeln('Press keys, press "q" to end. ');
  Repeat
    K:=GetKeyEvent;
    K:=TranslateKeyEvent(K);
    If IsFunctionKey(K) then
      Writeln('Got function key : ',KeyEventToString(K))
    else
      Writeln('not a function key. ');
  Until (GetKeyEventChar(K)='q');
  DoneKeyBoard;
end.

```

59.7.12 KeyEventToString

Synopsis: Return a string describing the key event.

Declaration: `function KeyEventToString(KeyEvent: TKeyEvent) : string`

Visibility: default

Description: `KeyEventToString` translates the key event in `KeyEvent` to a human-readable description of the pressed key. It will use the constants described in the constants section to do so.

For an example, see most other functions.

Errors: If an unknown key is passed, the scancode is returned, prefixed with the `SScanCode` string.

See also: [FunctionKeyName \(982\)](#), [ShiftStateToString \(990\)](#)

59.7.13 KeyPressed

Synopsis: Check event queue for key press.

Declaration: `function KeyPressed : Boolean`

Visibility: default

Description: `KeyPressed` checks the keyboard event queue to see whether a key event is present, and returns `True` if a key event is available. This function simply calls [PollKeyEvent \(988\)](#) and checks for a valid result.

Errors: None.

See also: [PollKeyEvent \(988\)](#), [GetKeyEvent \(983\)](#)

59.7.14 PollKeyEvent

Synopsis: Get next key event, but does not wait.

Declaration: `function PollKeyEvent : TKeyEvent`

Visibility: default

Description: `PollKeyEvent` checks whether a key event is available, and returns it if one is found. If no event is pending, it returns 0.

Note that this does not remove the key from the pending keys. The key should still be retrieved from the pending key events list with the `GetKeyEvent` (983) function.

Errors: None.

See also: `PutKeyEvent` (989), `GetKeyEvent` (983)

Listing: `./examples/kbdex/ex4.pp`

```

program example4;

{ This program demonstrates the PollKeyEvent function }

uses keyboard;

Var
  K : TKeyEvent;

begin
  InitKeyBoard;
  Writeln('Press keys, press "q" to end. ');
  Repeat
    K:=PollKeyEvent;
    If k<>0 then
      begin
        K:=GetKeyEvent;
        K:=TranslateKeyEvent(K);
        writeln;
        Writeln('Got key : ',KeyEventToString(K));
      end
    else
      write(' ');
    Until (GetKeyEventChar(K)='q');
  DoneKeyBoard;
end.

```

59.7.15 PollShiftStateEvent

Synopsis: Check current shift state.

Declaration: `function PollShiftStateEvent : TKeyEvent`

Visibility: default

Description: `PollShiftStateEvent` returns the current shiftstate in a keyevent. This will return 0 if there is no key event pending.

Errors: None.

See also: [PollKeyEvent \(988\)](#), [GetKeyEvent \(983\)](#)

Listing: ./examples/kbdex/ex6.pp

```

program example6;

{ This program demonstrates the PollShiftStateEvent function }

uses keyboard;

Var
  K : TKeyEvent;

begin
  InitKeyBoard;
  WriteLn('Press keys, press "q" to end. ');
  Repeat
    K:=PollKeyEvent;
    If k<>0 then
      begin
        K:=PollShiftStateEvent;
        WriteLn('Got shift state : ',ShiftStateToString(K,False));
        // Consume the key.
        K:=GetKeyEvent;
        K:=TranslateKeyEvent(K);
      end
    { else
      write('. ');
    }
    Until (GetKeyEventChar(K)='q');
    DoneKeyBoard;
  end.

```

59.7.16 PutKeyEvent

Synopsis: Put a key event in the event queue.

Declaration: `procedure PutKeyEvent (KeyEvent : TKeyEvent)`

Visibility: default

Description: `PutKeyEvent` adds the given `KeyEvent` to the input queue. Please note that depending on the implementation this can hold only one value, i.e. when calling `PutKeyEvent` multiple times, only the last pushed key will be remembered.

Errors: None

See also: [PollKeyEvent \(988\)](#), [GetKeyEvent \(983\)](#)

Listing: ./examples/kbdex/ex5.pp

```

program example5;

{ This program demonstrates the PutKeyEvent function }

uses keyboard;

Var
  K,k2 : TKeyEvent;

```

```

begin
  InitKeyBoard;
  Writeln('Press keys, press "q" to end.');
```

K2:=0;

Repeat

```

  K:=GetKeyEvent;
  If k<>0 then
    begin
      if (k2 mod 2)=0 then
        K2:=K+1
      else
        K2:=0;
      K:=TranslateKeyEvent(K);
      Writeln('Got key : ',KeyEventToString(K));
      if (K2<>0) then
        begin
          PutKeyEvent(k2);
          K2:=TranslateKeyEvent(K2);
          Writeln('Put key : ',KeyEventToString(K2))
        end
      end
    Until (GetKeyEventChar(K)='q');
  DoneKeyBoard;
end.
```

59.7.17 SetKeyboardDriver

Synopsis: Set a new keyboard driver.

Declaration: `function SetKeyboardDriver(const Driver: TKeyboardDriver) : Boolean`

Visibility: default

Description: `SetKeyBoardDriver` sets the keyboard driver to `Driver`, if the current keyboard driver is not yet initialized. If the current keyboard driver is initialized, then `SetKeyboardDriver` does nothing. Before setting the driver, the currently active driver should be disabled with a call to `DoneKeyboard` (981).

The function returns `True` if the driver was set, `False` if not.

For more information on setting the keyboard driver, see `kbddriver` (971).

Errors: None.

See also: `GetKeyboardDriver` (982), `DoneKeyboard` (981)

59.7.18 ShiftStateToString

Synopsis: Return description of key event shift state.

Declaration: `function ShiftStateToString(KeyEvent: TKeyEvent; UseLeftRight: Boolean) : string`

Visibility: default

Description: `ShiftStateToString` returns a string description of the shift state of the key event `KeyEvent`. This can be an empty string.

The shift state is described using the strings in the `SShift` constant.

For an example, see `PollShiftStateEvent` (988).

Errors: None.

See also: `FunctionKeyName` (982), `KeyEventToString` (987)

59.7.19 TranslateKeyEvent

Synopsis: Translate raw event to ascii key event.

Declaration: `function TranslateKeyEvent (KeyEvent: TKeyEvent) : TKeyEvent`

Visibility: default

Description: `TranslateKeyEvent` performs ASCII translation of the `KeyEvent`. It translates a physical key to a function key if the key is a function key, and translates the physical key to the ordinal of the ascii character if there is an equivalent character key.

For an example, see `GetKeyEvent` (983)

Errors: None.

See also: `TranslateKeyEventUnicode` (991)

59.7.20 TranslateKeyEventUnicode

Synopsis: Translate raw event to UNICODE key event.

Declaration: `function TranslateKeyEventUnicode (KeyEvent: TKeyEvent) : TKeyEvent`

Visibility: default

Description: `TranslateKeyEventUnicode` performs Unicode translation of the `KeyEvent`. It is not yet implemented for all platforms.

Errors: If the function is not yet implemented, then the `ErrorCode` of the `system` unit will be set to `errKbdNotImplemented`

See also: `TranslateKeyEvent` (991)

59.8 TKeyboardDriver

```
TKeyboardDriver = record
  InitDriver : procedure;
  DoneDriver :
  procedure;
  GetKeyEvent : function : TKeyEvent;
  PollKeyEvent
  : function : TKeyEvent;
  GetShiftState : function : Byte;
  TranslateKeyEvent
  : function (KeyEvent: TKeyEvent) : TKeyEvent;
```



```

    TranslateKeyEventUnicode
    : function(KeyEvent: TKeyEvent) : TKeyEvent;
end

```

The `TKeyboardDriver` record can be used to install a custom keyboard driver with the `SetKeyboardDriver` (990) function.

The various fields correspond to the different functions of the keyboard unit interface. For more information about this record see `kbddriver` (971)

59.9 TKeyRecord

```

TKeyRecord = packed record
    Flags : Byte;
    ShiftState : Byte;
    KeyCode : Word;
end

```

The structure of a `TKeyRecord` structure is explained in the following table:

Table 59.4: Structure of `TKeyRecord`

Field	Meaning
<code>KeyCode</code>	Depending on <code>flags</code> either the physical representation of a key (under DOS scancode, ASCII code pair), or the
<code>ShiftState</code>	Shift-state when this key was pressed (or shortly after)
<code>Flags</code>	Determine how to interpret <code>KeyCode</code>

The shift-state can be checked using the various shift-state constants, and the flags in the last byte can be checked using one of the `kbASCII`, `kbUnicode`, `kbFnKey`, `kbPhys`, `kbReleased` constants.

If there are two keys returning the same char-code, there's no way to find out which one was pressed (Gray+ and Simple+). If it needs to be known which was pressed, the untranslated keycodes must be used, but these are system dependent. System dependent constants may be defined to cover those, with possibly having the same name (but different value).

59.10 TTreeElement

```

TTreeElement = record
    Next : PTreeElement;
    Parent : PTreeElement
    ;
    Child : PTreeElement;
    CanBeTerminal : Boolean;
    char : Byte
    ;
    ScanValue : Byte;
    CharValue : Byte;
    SpecialHandler : Tprocedure
    ;
end

```

`TTreeElement` is used to describe key scancode sequences, and is used to handle special key combinations in `AddSpecialSequence` (??) on UNIX platforms. There should be no need to handle records of this type.

Chapter 60

Reference for unit 'lineinfo'

60.1 Used units

Table 60.1: Used units by unit 'lineinfo'

Name	Page
System	1335

60.2 Overview

The `lineinfo` provides a routine that reads the debug information of an executable (if any exists) and returns source code information about this address. It works with `Stabs` debug information. Note that this unit is not thread-safe, and that its behaviour is undefined if multiple threads try to write a backtrace at the same time.

For DWARF debug information, the `Infodwrf` ([1027](#)) unit must be used.

60.3 Constants, types and variables

60.3.1 Types

`CodePointer = Pointer`

`CodePointer` is added for 16-bit dos compatibility.

60.3.2 Variables

`AllowReuseOfLineInfoData : Boolean = True`

`AllowReuseOfLineInfoData` can be set to `True` to keep the last opened file open. When regularly creating backtraces (e.g. in a program log), this will significantly speed up operations.

60.4 Procedures and functions

60.4.1 CloseStabs

Synopsis: Close stabs info file descriptor.

Declaration: `procedure CloseStabs`

Visibility: default

Description: `CloseStabs` will close the file descriptor that was used to read STABS debug information. This is useful if `AllowReuseOfLineInfoData` (994) is used to cache STABS information.

Errors: None.

See also: `AllowReuseOfLineInfoData` (994)

60.4.2 GetLineInfo

Synopsis: Return source line information about an address.

Declaration: `function GetLineInfo(addr: PtrUInt; var func: string;
var source: string; var line: LongInt) : Boolean`

Visibility: default

Description: `GetLineInfo` returns source line information about the address `addr`. It searches this information in the stabs debugging information found in the binary: If the file was compiled without debug information, nothing will be returned. Upon successful retrieval of the debug information, `True` is returned, and the `func` parameter is filled with the name of the function in which the address is located. The `source` parameter contains the name of the file in which the function was implemented, and `line` contains the line number in the source file for `addr`.

Errors: If no debug information is found, `False` is returned.

60.4.3 StabBackTraceStr

Synopsis: Get a backtrace from an address.

Declaration: `function StabBackTraceStr(addr: CodePointer) : string`

Visibility: default

Description: `StabBackTraceStr` returns a backtrace from a memory address `Addr`.

This is the actual callback for the backtrace handler `System.BackTraceStrFunc` (994).

Errors: None.

See also: `GetLineInfo` (995)

Chapter 61

Reference for unit 'Linux'

61.1 Used units

Table 61.1: Used units by unit 'Linux'

Name	Page
BaseUnix	155
System	1335
unixtype	2229

61.2 Overview

The linux unit contains Linux specific operating system calls.

The platform independent functionality of the FPC 1.0.X version of the linux unit has been split out over the UNIX ([2189](#)), baseunix ([155](#)) and unixutil ([2245](#)) units.

The X86-specific parts have been moved to the X86 ([2326](#)) unit.

61.3 Constants, types and variables

61.3.1 Constants

`CAP_AUDIT_CONTROL = 30`

Allow manipulation of kernel auditing features.

`CAP_AUDIT_WRITE = 29`

Allow writing to kernel audit log.

`CAP_CHOWN = 0`

Perform chown operation.

`CAP_DAC_OVERRIDE = 1`

Bypass file operation (rwx) checks.

`CAP_DAC_READ_SEARCH = 2`

Bypass file read-only operation checks.

`CAP_FOWNER = 3`

Bypass owner ID checks.

`CAP_FSETID = 4`

Do not clear SUID/GUID bits on modified files.

`CAP_FS_MASK = 0xf`

?

`CAP_IPC_LOCK = 14`

Allow memory locking calls.

`CAP_IPC_OWNER = 15`

Bypass permission checks on IPC operations.

`CAP_KILL = 5`

Bypass permission checks for sending signals.

`CAP_LEASE = 28`

Allow file leases.

`CAP_LINUX_IMMUTABLE = 9`

Allow setting ext2 file attributes.

`CAP_MKNOD = 27`

Allow creation of special files through mknod calls.

`CAP_NET_ADMIN = 12`

Allow network operations (e.g. setting socket options).

`CAP_NET_BIND_SERVICE = 10`

Allow binding to ports less than 1024.

`CAP_NET_BROADCAST = 11`

Allow socket broadcast operations.

`CAP_NET_RAW = 13`

Allow use of RAW and PACKET sockets.

`CAP_SETGID = 6`

Allow GID manipulations.

`CAP_SETPCAP = 8`

Allow to set other process' capabilities.

`CAP_SETUID = 7`

Allow process ID manipulations.

`CAP_SYS_ADMIN = 21`

Allow various system administration calls.

`CAP_SYS_BOOT = 22`

Allow reboot calls.

`CAP_SYS_CHROOT = 18`

Allow chroot calls.

`CAP_SYS_MODULE = 16`

Allow loading/unloading of kernel modules.

`CAP_SYS_NICE = 23`

Allowing raising process and thread priorities.

`CAP_SYS_PACCT = 20`

Allow acct calls.

`CAP_SYS_PTRACE = 19`

Allow ptrace calls.

`CAP_SYS_RAWIO = 17`

Allow raw I/O port operations.

`CAP_SYS_RESOURCE = 24`

Allow use of special resources or raising of resource limits.

`CAP_SYS_TIME = 25`

Allow system or real-time clock modification.

`CAP_SYS_TTY_CONFIG = 26`

Allow vhangup calls.

`CLOCKS_MASK = CLOCK_REALTIME or CLOCK_MONOTONIC`

Mask for supported clocks.

`CLOCKS_MONO = CLOCK_MONOTONIC`

Monotonic clocks mask.

`CLOCK_MONOTONIC = 1`

Monotonic system time since some undetermined start point. Can change if time is set.

`CLOCK_MONOTONIC_COARSE = 6`

Less precise (but faster) version of `CLOCK_MONOTONIC`.

`CLOCK_MONOTONIC_RAW = 4`

Like `CLOCK_MONOTONIC`, not subject to NTP adjustments.

`CLOCK_PROCESS_CPUTIME_ID = 2`

Process-specific high-resolution timer from the CPU.

`CLOCK_REALTIME = 0`

System wide real-time clock. Can only be set by root.

`CLOCK_REALTIME_COARSE = 5`

Less precise (but faster) version of `CLOCK_REALTIME`.

`CLOCK_SGI_CYCLE = 10`

High resolution timer.

`CLOCK_THREAD_CPUTIME_ID = 3`

Thread-specific high-resolution timer from the CPU.

`CLONE_CHILD_CLEARTID = $00200000`

Clone option: Erase child thread ID in child memory space when child exits.

`CLONE_CHILD_SETTID = $01000000`

Clone option: Store child thread ID in child memory.

`CLONE_DETACHED = $00400000`

Clone option: Start clone detached.

`CLONE_FILES = $00000400`

Clone (1015) option: open files shared between processes.

`CLONE_FS = $00000200`

Clone (1015) option: fs info shared between processes.

`CLONE_NEWNS = $00020000`

Clone options: Start child in new (file system) namespace.

`CLONE_PARENT = $00008000`

Clone options: Set child parent to parent of calling process.

`CLONE_PARENT_SETTID = $00100000`

Clone option: Store child thread ID in memory in both parent and child.

`CLONE_PID = $00001000`

Clone (1015) option: PID shared between processes.

`CLONE_PTRACE = $00002000`

Clone options: if parent is traced, trace child also.

`CLONE_SETTLS = $00080000`

Clone option: The newtls parameter is the TLS descriptor of the child.

`CLONE_SIGHAND = $00000800`

Clone (1015) option: signal handlers shared between processes.

`CLONE_STOPPED = $02000000`

Clone option: Start child in stopped state.

`CLONE_SYSVSEM = $00040000`

Clone option: Caller and child share the same semaphore undo values.

`CLONE_THREAD = $00010000`

Clone options: Set child in thread group of calling process.

`CLONE_UNTRACED = $00800000`

Clone option: Do not allow a ptrace call on this clone.

`CLONE_VFORK = $00004000`

Clone options: suspend parent till child execs.

`CLONE_VM = $00000100`

Clone (1015) option: VM shared between processes.

`CSIGNAL = $000000ff`

Clone (1015) option: Signal mask to be sent at exit.

`EPOLLERR = $08`

`event_wait` error condition on file descriptor.

`EPOLLET = $80000000`

Set `event_wait` edge trigger behaviour on file descriptor.

`EPOLLHUP = $10`

`event_wait` hang up event.

`EPOLLIN = $01`

`event_wait` input file descriptor ready event.

`EPOLLONESHOT = $40000000`

Set single-shot behaviour on `epoll_wait`.

`EPOLLOUT = $04`

`event_wait` output file descriptor ready event.

`EPOLLPRI = $02`

`event_wait` high priority data available on input file descriptor.

`EPOLL_CTL_ADD = 1`

Add filedescriptor to list of events.

`EPOLL_CTL_DEL = 2`

Delete event for filedescriptor.

`EPOLL_CTL_MOD = 3`

Modify event for filedescriptor.

`FUTEX_CMP_REQUEUE = 4`

Futex option: requeue waiting processes on other futex, but check it's value first.

`FUTEX_FD = 2`

Futex option: Associate file descriptor with futex.

`FUTEX_LOCK_PI = 6`

Futex option: Undocumented.

`FUTEX_OP_ADD = 1`

Futex operation: Undocumented.

`FUTEX_OP_ANDN = 3`

Futex operation: Undocumented.

`FUTEX_OP_CMP_EQ = 0`

Futex operation: Undocumented.

`FUTEX_OP_CMP_GE = 5`

Futex operation: Undocumented.

`FUTEX_OP_CMP_GT = 4`

Futex operation: Undocumented.

`FUTEX_OP_CMP_LE = 3`

Futex operation: Undocumented.

`FUTEX_OP_CMP_LT = 2`

Futex operation: Undocumented.

FUTEX_OP_CMP_NE = 1

Futex operation: Undocumented.

FUTEX_OP_OPARG_SHIFT = 8

Futex operation: Undocumented.

FUTEX_OP_OR = 2

Futex operation: Undocumented.

FUTEX_OP_SET = 0

Futex operation: Undocumented.

FUTEX_OP_XOR = 4

Futex operation: Undocumented.

FUTEX_REQUEUE = 3

Futex option: requeue waiting processes on other futex.

FUTEX_TRYLOCK_PI = 8

Futex option: Undocumented.

FUTEX_UNLOCK_PI = 7

Futex option: Undocumented.

FUTEX_WAIT = 0

Futex option: Wait on futex till wake call arrives.

FUTEX_WAKE = 1

Futex option: wakes any waiting processes on this futex.

FUTEX_WAKE_OP = 5

Futex option: Undocumented.

GIO_CMAP = \$4B70

IOCTL: Get color palette on VGA+.

GIO_FONT = \$4B60

IOCTL: Get font in expanded form.

GIO_FONTX = \$4B6B

IOCTL: Get font in consolefontdesc record.

GIO_SCRNMAP = \$4B40

IOCTL: get screen mapping from kernel.

GIO_UNIMAP = \$4B66

IOCTL: get unicode-to-font mapping from kernel.

GIO_UNISCRNMAP = \$4B69

IOCTL: get full Unicode screen mapping.

IN_ACCESS = \$00000001

Data was read from file.

IN_ALL_EVENTS = IN_ACCESS or IN_MODIFY or IN_ATTRIB or IN_CLOSE or
IN_OPEN or IN_MOVE or IN_CREATE or IN_DELETE or IN_DELETE_SELF or
IN_MOVE_SELF

All possible events OR-ed together.

IN_ATTRIB = \$00000004

File attributes changed.

IN_CLOEXEC = &02000000

IN_CLOEXEC can be set to indicate that the inotify file handle must be closed on exec.

IN_CLOSE = IN_CLOSE_WRITE or IN_CLOSE_NOWRITE

File was closed (read or write).

IN_CLOSE_NOWRITE = \$00000010

File opened for read was closed.

IN_CLOSE_WRITE = \$00000008

File opened for write was closed.

IN_CREATE = \$00000100

A file was created in the directory.

IN_DELETE = \$00000200

A file was deleted from the directory.

`IN_DELETE_SELF = $00000400`

Directory or file under observation was deleted.

`IN_DONT_FOLLOW = $02000000`

Do not follow symlinks.

`IN_IGNORED = $00008000`

Watch was ignored (removed). Only reported.

`IN_ISDIR = $40000000`

Event subject is a directory (reported only).

`IN_MASK_ADD = $20000000`

Add events to existing watch (OR-ing the sets) if one exists.

`IN_MODIFY = $00000002`

Data was written to file.

`IN_MOVE = IN_MOVED_FROM or IN_MOVED_TO`

File was moved (in or out of directory).

`IN_MOVED_FROM = $00000040`

File was moved away from watched directory.

`IN_MOVED_TO = $00000080`

File was moved into watched directory.

`IN_MOVE_SELF = $00000800`

Directory or file under observation was moved.

`IN_NONBLOCK = &00004000`

`IN_NONBLOCK` can be set to indicate that the inotify file handle should not block read operations.

`IN_ONESHOT = $80000000`

Only report one event, then remove the watch.

`IN_ONLYDIR = $01000000`

Only watch filename if it is a directory.

IN_OPEN = \$00000020

File was opened.

IN_Q_OVERFLOW = \$00004000

Queue overflowed. Only reported.

IN_UNMOUNT = \$00002000

File system on which file resides was unmounted. Only reported.

KB_101 = 2

IOCTL: Keyboard types: 101 keys.

KB_84 = 1

IOCTL: Keyboard types: 84 keys.

KB_OTHER = 3

IOCTL: Keyboard types: other type.

KDADDIO = \$4B34

IOCTL: add i/o port as valid.

KDDELIO = \$4B35

IOCTL: delete i/o port as valid.

KDDISABIO = \$4B37

IOCTL: disable i/o to video board.

KDENABIO = \$4B36

IOCTL: enable i/o to video board.

KDFONTOP = \$4B72

IOCTL: font operations.

KDGETKEYCODE = \$4B4C

IOCTL: read kernel keycode table entry.

KDGETLED = \$4B31

IOCTL: return current led state.

KDGETMODE = \$4B3B

IOCTL: get current mode.

KDGKBDIACR = \$4B4A

IOCTL: read kernel accent table.

KDGKBTYPE = \$4B33

IOCTL: get keyboard type.

KDMAPDISP = \$4B3C

IOCTL: map display into address space.

KDMKTONE = \$4B30

IOCTL: generate tone.

KDSETKEYCODE = \$4B4D

IOCTL: write kernel keycode table entry.

KDSETLED = \$4B32

IOCTL: set led state.

KDSETMODE = \$4B3A

IOCTL: set text/graphics mode.

KDSIGACCEPT = \$4B4E

IOCTL: accept kbd generated signals.

KDSKBDIACR = \$4B4B

IOCTL: write kernel accent table.

KDUNMAPDISP = \$4B3D

IOCTL: unmap display from address space.

KD_GRAPHICS = 1

IOCTL: Tty modes: graphics mode.

KD_TEXT = 0

IOCTL: Tty modes: Text mode.

KD_TEXT0 = 2

IOCTL: Tty modes: Text mode (obsolete).

KD_TEXT1 = 3

IOCTL: Tty modes: Text mode (obsolete).

KIOCSOUND = \$4B2F

IOCTL: start/stop sound generation (0 for off).

LED_CAP = 4

IOCTL: LED_CAP : caps lock led.

LED_NUM = 2

IOCTL: LED_SCR : Num lock led.

LED_SCR = 1

IOCTL: LED_SCR : scroll lock led.

LINUX_CAPABILITY_VERSION = \$19980330

Current capability version in use by kernel.

MAP_DENYWRITE = \$800

Read-only.

MAP_EXECUTABLE = \$1000

Memory area is marked as executable.

MAP_GROWSDOWN = \$100

Memory map grows down, like stack.

MAP_LOCKED = \$2000

Memory pages are locked.

MAP_NORESERVE = \$4000

Do not check for reservations.

MAX_CLOCKS = 16

Maximum number of clocks in the system.

MODIFY_LDT_CONTENTS_CODE = 2

Modify_ldt option: Undocumented.

MODIFY_LDT_CONTENTS_DATA = 0

Modify_ldt option: Undocumented.

MODIFY_LDT_CONTENTS_STACK = 1

Modify_ldt option: Undocumented.

O_CLOEXEC = \$80000

Close on exec flag: close file handle on exec call.

PIO_CMAP = \$4B71

IOCTL: Set color palette on VGA+.

PIO_FONT = \$4B61

IOCTL: Use font in expanded form.

PIO_FONTRESET = \$4B6D

IOCTL: Reset to default font.

PIO_FONTX = \$4B6C

IOCTL: Set font in consolefontdesc record.

PIO_SCRNMAP = \$4B41

IOCTL: put screen mapping table in kernel.

PIO_UNIMAP = \$4B67

IOCTL: put unicode-to-font mapping in kernel.

PIO_UNIMAPCLR = \$4B68

IOCTL: clear table, possibly advise hash algorithm.

PIO_UNISCRNMAP = \$4B6A

IOCTL: set full Unicode screen mapping.

POLLMSG = \$0400

Unused in Linux.

POLLRDHUP = \$2000

Peer Shutdown/closed writing half of connection.

POLLREMOVE = \$1000

Undocumented Linux extension of Poll.

SPLICE_F_GIFT = 8

Pages spliced in are a gift.

SPLICE_F_MORE = 4

Expect more data.

SPLICE_F_MOVE = 1

Move pages instead of copying.

SPLICE_F_NONBLOCK = 2

Don't block on pipe splicing operations.

STATX_ALL = \$00000fff

STATX_ATIME = \$00000020

STATX_ATTR_APPEND = \$00000020

STATX_ATTR_AUTOMOUNT = \$00001000

STATX_ATTR_COMPRESSED = \$00000004

STATX_ATTR_ENCRYPTED = \$00000800

STATX_ATTR_IMMUTABLE = \$00000010

STATX_ATTR_NODUMP = \$00000040

STATX_BASIC_STATS = \$000007ff

STATX_BLOCKS = \$00000400

STATX_BTIME = \$00000800

STATX_CTIME = \$00000080

STATX_GID = \$00000010

STATX_INO = \$00000100

STATX_MODE = \$00000002

STATX_MTIME = \$00000040

STATX_NLINK = \$00000004

STATX_SIZE = \$00000200

STATX_TYPE = \$00000001

STATX_UID = \$00000008

STATX__RESERVED = \$80000000

SYNC_FILE_RANGE_WAIT_AFTER = 4

Wait upon write-out of specified pages in the range after performing any write.

SYNC_FILE_RANGE_WAIT_BEFORE = 1

Wait for write-out of previously-submitted specified pages before writing more data.

SYNC_FILE_RANGE_WRITE = 2

Initiate write of all dirty pages in the specified range.

UD_CONTENTS_CODE = MODIFY_LDT_CONTENTS_CODE shl 1

TLS segment descriptor: Undocumented.

UD_CONTENTS_DATA = MODIFY_LDT_CONTENTS_DATA shl 1

TLS segment descriptor: Undocumented.

```
UD_CONTENTS_STACK = MODIFY_LDT_CONTENTS_STACK shl 1
```

TLS segment descriptor: Undocumented.

```
UD_LIMIT_IN_PAGES = $10
```

TLS segment descriptor: Undocumented.

```
UD_LM = $80
```

TLS segment descriptor: Undocumented.

```
UD_READ_EXEC_ONLY = $08
```

TLS segment descriptor: Undocumented.

```
UD_SEG_32BIT = $01
```

TLS segment descriptor : Undocumented.

```
UD_SEG_NOT_PRESENT = $20
```

TLS segment descriptor: Undocumented.

```
UD_USEABLE = $40
```

TLS segment descriptor: Undocumented.

61.3.2 Types

```
clockid_t = cint
```

Clock id type.

```
EPoll_Data = record
case Integer of
0: (
    ptr : pointer;
);
1: (
    fd : cint;
);
2: (
    u32 : cuint;
);
3: (
    u64 : cuint64;
);
end
```

Data structure used in EPOLL_IOCTL call.

```
kernel_time64_t = clonglong
```

```
PEPoll_Data = ^EPoll_Data
```

Pointer to EPoll_Data (1013) record.

```
PEpoll_Event = ^EPoll_Event
```

Pointer to EPoll_Event (1024) type.

```
Pinotify_event = ^inotify_event
```

Pointer to inotify_event (1024) structure.

```
pkernel_timespec = ^kernel_timespec
```

```
pstatx = ^tstatx
```

```
pstatx_timestamp = ^statx_timestamp
```

```
PSysInfo = ^TSysInfo
```

Pointer to TSysInfo (1025) record.

```
Puser_cap_data = ^user_cap_data
```

Pointer to user_cap_data (1026) record.

```
Puser_cap_header = ^user_cap_header
```

Pointer to user_cap_header (1026) record.

```
PUser_Desc = ^user_desc
```

PUser_Desc is a pointer to the user_desc (1026) type.

```
TCloneFunc = function(args: pointer) : LongInt
```

Clone function prototype.

```
TEPoll_Data = EPoll_Data
```

Alias for EPoll_Data (1013) type.

```
TEPoll_Event = EPoll_Event
```

Alias for EPoll_Event ([1024](#)) type.

`tkernel_timespecs` = Array[0..1] of `kernel_timespec`

`TUser_Desc` = `user_desc`

`TUser_Desc` is an alias for the `user_desc` ([1026](#)) type.

`__s16` = `SmallInt`

`__s32` = `LongInt`

`__s64` = `Int64`

`__u16` = `Word`

`__u32` = `DWord`

`__u64` = `QWord`

61.4 Procedures and functions

61.4.1 capget

Synopsis: Return the capabilities for the indicated thread.

Declaration: `function capget(header: Puser_cap_header; data: Puser_cap_data) : cint`

Visibility: default

Description: `capget` returns the capabilities of the indicated thread in `header`. The thread is identified by the process ID, or -1 for all caller (and child) process ID's.

Refer to the Linux man pages (7 capabilities) for more info.

Errors: On success, zero is returned, on error -1 is returned, and `fperno` is set to the error.

See also: `capset` ([1014](#))

61.4.2 capset

Synopsis: Set the capabilities for the indicated thread.

Declaration: `function capset(header: Puser_cap_header; data: Puser_cap_data) : cint`

Visibility: default

Description: `capset` sets the capabilities of the indicated thread in `header`. The thread is identified by the process ID, or -1 for all caller (and child) process ID's.

Refer to the Linux man pages (7 capabilities) for more info.

Errors: On success, zero is returned, on error -1 is returned, and `fperno` is set to the error.

See also: `capget` ([1014](#))

61.4.3 clock_getres

Synopsis: Get clock resolution.

Declaration: `function clock_getres(clk_id: clockid_t; res: timespec) : cint`

Visibility: default

Description: `clock_getres` returns the resolution of the clock specified in `clk_id` in the `res` structure. It can be `Nil`. if the clock exists and the resolution can be retrieved, 0 is returned.

Errors: On Error, -1 is returned. `fpgeterrno` can be used to get more detailed error information.

See also: `clock_gettime` ([1015](#)), `clock_settime` ([1015](#))

61.4.4 clock_gettime

Synopsis: Get the time of a clock.

Declaration: `function clock_gettime(clk_id: clockid_t; tp: timespec) : cint`

Visibility: default

Description: `clock_gettime` returns the current time of the clock specified in `clk_id` in the `tp` structure. If the clock exists and the time can be retrieved, 0 is returned.

Errors: On Error, -1 is returned. `fpgeterrno` can be used to get more detailed error information.

See also: `clock_getres` ([1015](#)), `clock_settime` ([1015](#))

61.4.5 clock_settime

Synopsis: Set the time of a clock.

Declaration: `function clock_settime(clk_id: clockid_t; tp: timespec) : cint`

Visibility: default

Description: `clock_settime` sets the current time of the clock specified in `clk_id`. The time is specified in the `tp` structure. If the clock exists and the time can be retrieved, 0 is returned. The resolution is truncated to the resolution supported by the specified clock. Note that not all clocks can be set.

Errors: On Error, -1 is returned. `fpgeterrno` can be used to get more detailed error information.

See also: `clock_getres` ([1015](#)), `clock_gettime` ([1015](#))

61.4.6 clone

Synopsis: Clone current process (create new thread).

Declaration: `function clone(func: TCloneFunc; sp: pointer; flags: LongInt; args: pointer) : LongInt`

Visibility: default

Description: `Clone` creates a child process which is a copy of the parent process, just like `FpFork` (204) does. In difference with `Fork`, however, the child process shares some parts of it's execution context with its parent, so it is suitable for the implementation of threads: many instances of a program that share the same memory.

When the child process is created, it starts executing the function `Func`, and passes it `Args`. The return value of `Func` is either the explicit return value of the function, or the exit code of the child process.

The `sp` pointer points to the memory reserved as stack space for the child process. This address should be the top of the memory block to be used as stack.

The `Flags` determine the behaviour of the `Clone` call. The low byte of the `Flags` contains the number of the signal that will be sent to the parent when the child dies. This may be bitwise OR'ed with the following constants:

CLONE_VM Parent and child share the same memory space, including memory (un)mapped with subsequent `mmap` calls.

CLONE_FS Parent and child have the same view of the file system; the `chroot`, `chdir` and `umask` calls affect both processes.

CLONE_FILES the file descriptor table of parent and child is shared.

CLONE_SIGHAND the parent and child share the same table of signal handlers. The signal masks are different, though.

CLONE_PID Parent and child have the same process ID.

`Clone` returns the process ID in the parent process, and -1 if an error occurred.

Errors: On error, -1 is returned to the parent, and no child is created.

sys_eagain Too many processes are running.

sys_enomem Not enough memory to create child process.

See also: `#rtl.baseunix.FpFork` (204)

61.4.7 `epoll_create`

Synopsis: Create new `epoll` file descriptor.

Declaration: `function epoll_create(size: cint) : cint`

Visibility: default

Description: `epoll_create` creates a new `epoll` file descriptor. The `size` argument indicates to the kernel approximately how many structures should be allocated, but is by no means an upper limit.

On success, a file descriptor is returned that can be used in subsequent `epoll_ctl` (1017) or `epoll_wait` (1017) calls, and should be closed using the `fpClose` (197) call.

Errors: On error, -1 is returned, and `errno` (207) is set.

See also: `epoll_ctl` (1017), `epoll_wait` (1017), `fpClose` (197)

61.4.8 `epoll_ctl`

Synopsis: Modify an epoll file descriptor.

Declaration: `function epoll_ctl(epfd: cint; op: cint; fd: cint; event: PEpoll_Event)
: cint`

Visibility: default

Description: `epoll_ctl` performs the `op` operation on epoll file descriptor `epfd`. The operation will be monitored on file descriptor `fd`, and is optionally controlled by `event`.

`op` can be one of the following values:

EPOLL_CTL_ADD Add filedescriptor to list of events.

EPOLL_CTL_MOD Modify event for filedescriptor.

EPOLL_CTL_DEL Delete event for filedescriptor.

The `events` field in `event_data` is a bitmask of one or more of the following values:

EPOLLIN The file is ready for read operations

EPOLLOUT The file is ready for write operations.

EPOLLPRI Urgent data is available for read operations.

EPOLLERR An error condition is signaled on the file descriptor.

EPOLLHUP A Hang up happened on the file descriptor.

EPOLLET Set the Edge Triggered behaviour for the file descriptor.

EPOLLONESHOT Set One-Shot behaviour for the file descriptor. The event will be triggered only once.

Errors: On error -1 is returned, and `errno` is set accordingly.

See also: `epoll_create` ([1016](#)), `epoll_wait` ([1017](#)), `fpClose` ([197](#))

61.4.9 `epoll_wait`

Synopsis: Wait for an event on an epoll file descriptor.

Declaration: `function epoll_wait(epfd: cint; events: PEpoll_Event; maxevents: cint;
timeout: cint) : cint`

Visibility: default

Description: `epoll_wait` waits for `timeout` milliseconds for an event to occur on epoll file descriptor `epfd`. If `timeout` is -1, it waits indefinitely, if `timeout` is zero, it does not wait, but returns immediately, even if no events were detected.

On return, data for at most `maxevents` will be returned in the memory pointed to by `events`. The function returns the number of file descriptors for which events were reported. This can be zero if the timeout was reached.

Errors: On error -1 is returned, and `errno` is set accordingly.

See also: `epoll_create` ([1016](#)), `epoll_ctl` ([1017](#)), `fpClose` ([197](#))

61.4.10 fdatasync

Synopsis: Synchronize the data in memory with the data on storage device.

Declaration: `function fdatasync(fd: cint) : cint`

Visibility: default

Description: `fdatasync` does the same as `ffsync` but does not flush the metadata, unless it is vital to the correct reading/writing of the file. In practice, this means that unless the file size changed, the file metadata will not be synced.

See also: `#rtl.unix.fsync` ([2189](#))

61.4.11 futex

Synopsis: Perform a futex operation.

Declaration: `function futex(uaddr: pcint; op: cint; val: cint; timeout: ptimespec;
addr2: pcint; val3: cint) : cint`
`function futex(var uaddr; op: cint; val: cint; timeout: ptimespec;
var addr2; val3: cint) : cint`
`function futex(var uaddr; op: cint; val: cint; var timeout: TTimeSpec;
var addr2; val3: cint) : cint`
`function futex(uaddr: pcint; op: cint; val: cint; timeout: ptimespec)
: cint`
`function futex(var uaddr; op: cint; val: cint; timeout: ptimespec)
: cint`
`function futex(var uaddr; op: cint; val: cint; var timeout: TTimeSpec)
: cint`

Visibility: default

Description: `futex` performs an operation on a memory futex as described in the kernel manual page for `futex`. The mutex is located at `uaddr`, the operation `op` is one of the following constants:

FUTEX_WAITFutex option: Wait on futex till wake call arrives.

FUTEX_WAKEFutex option: Wait on futex till wake call arrives.

FUTEX_FDFutex option: Associate file descriptor with futex.

FUTEX_REQUEUEFutex option: requeue waiting processes on other futex.

FUTEX_CMP_REQUEUEFutex option: requeue waiting processes on other futex, but check it's value first.

The value to check for is indicated in `val`, and a timeout can be specified in `timeout`. The optional arguments `addr2` and `val3` are used only with the `FUTEX_REQUEUE` and `FUTEX_CMP_REQUEUE` operations.

In case of an error, -1 is return. All other return values must be interpreted according to the operation performed.

This call directly interfaces with the Linux kernel, more information can be found in the kernel manual pages.

Errors: On error, -1 is returned. Use `#rtl.baseunix.fpgeterrno` ([207](#)) to get the error code.

61.4.12 futex_op

Synopsis: Futex operation:.

Declaration: `function futex_op(op: cint; oparg: cint; cmp: cint; cmparg: cint) : cint`

Visibility: default

Description: `FUTEX_OP` Performs an operation on a futex:

```

FUTEX_OP := ((op and $F) shl 28) or
             ((cmp and $F) shl 24) or
             ((oparg and $FFF) shl 12)
             or (cmparg and $FFF);

```

61.4.13 futimens

Declaration: `function futimens(fd: cint; const times: tkernel_timespecs) : cint`

Visibility: default

61.4.14 inotify_add_watch

Synopsis: Add a watch to a notify file descriptor.

Declaration: `function inotify_add_watch(fd: cint; name: PChar; mask: cuint32) : cint`

Visibility: default

Description: `inotify_add_watch` can be used to add a watch to an initialized inotify file descriptor (`fd`). The file or directory to watch can be specified in the `name` parameter, and the events that must be reported can be specified in `mask`. The following flags can be specified:

IN_ACCESSData was read from file.

IN_MODIFYData was written to file.

IN_ATTRIBFile attributes changed.

IN_CLOSE_WRITEFile opened for write was closed

IN_CLOSE_NOWRITEFile opened for read was closed

IN_CLOSEFile was closed (read or write)

IN_OPENFile was opened

IN_MOVED_FROMFile was moved away from watched directory

IN_MOVED_TOFile was moved into watched directory

IN_MOVEFile was moved (in or out of directory)

IN_CREATEA file was created in the directory.

IN_DELETEA file was deleted from the directory.

IN_DELETE_SELFDirectory or file under observation was deleted.

IN_MOVE_SELFDirectory or file under observation was moved.

IN_ALL_EVENTSAll possible events OR-ed together.

These events can be OR-ed with some flags, controlling the behaviour of the watch:

IN_ONLYDIROnly watch filename if it is a directory.

IN_ISDIREvent occurred against directory.

IN_DONT_FOLLOWDo not follow symlinks

IN_MASK_ADDAdd events to existing watch (OR-ing the sets) if one exists.

IN_ONESHOTOnly report one event, then remove the watch.

On return, the function returns a watch descriptor, which will be reported in the `inotify_event` (1024) structure's `wd`.

Errors: On Error, -1 is returned. `fpgeterrno` can be used to get more detailed error information.

See also: `inotify_init` (1020), `inotify_init1` (1020), `inotify_rm_watch` (1020), `inotify_event` (1024)

61.4.15 `inotify_init`

Synopsis: Initialize a new `inotify` file descriptor.

Declaration: `function inotify_init : cint`

Visibility: default

Description: `inotify_init` initializes a new `INotify` file descriptor. No options can be specified. On return, the file descriptor is returned.

Errors: On Error, -1 is returned. `fpgeterrno` can be used to get more detailed error information

See also: `inotify_init1` (1020), `inotify_add_watch` (1019), `inotify_rm_watch` (1020)

61.4.16 `inotify_init1`

Synopsis: Initialize a new `inotify` file descriptor with extra options.

Declaration: `function inotify_init1(flags: cint) : cint`

Visibility: default

Description: `inotify_init1` initializes a new `INotify` file descriptor. The following options can be OR-ed and passed in flags:

IN_NONBLOCKDo not block on read.

IN_CLOEXECInotify close on exec flag.

Errors: On Error, -1 is returned. `fpgeterrno` can be used to get more detailed error information.

See also: `inotify_init` (1020), `inotify_add_watch` (1019), `inotify_rm_watch` (1020)

61.4.17 `inotify_rm_watch`

Synopsis: Remove watch from `Inotify` file descriptor.

Declaration: `function inotify_rm_watch(fd: cint; wd: cint) : cint`

Visibility: default

Description: `inotify_rm_watch` removes watch descriptor `wd` from `inotify` descriptor `fd`. On success, 0 is returned.

Errors: On Error, -1 is returned. `fpgeterrno` can be used to get more detailed error information.

See also: `inotify_init` (1020), `inotify_init1` (1020), `inotify_add_watch` (1019), `inotify_event` (1024)

61.4.18 `modify_ldt`

Declaration: `function modify_ldt(func: cint; p: pointer; bytecount: culong) : cint`

Visibility: default

61.4.19 `sched_yield`

Synopsis: Yield the processor to another thread.

Declaration: `procedure sched_yield`

Visibility: default

Description: `sched_yield` yields the processor to another thread. The current thread is put at the back of its queue. If there is only 1 thread in the application, the thread continues to run. The call always returns zero.

61.4.20 `setregid`

Synopsis: Set Real and Effective Group ID.

Declaration: `function setregid(rgid: uid_t; egid: uid_t) : cint`

Visibility: default

Description: `setregid` sets the real group ID to `rgid` and the effective group ID to `egid`. Passing a value of -1 tells the system not to change that value.

Errors: On Error, -1 is returned. `fpgeterrno` can be used to get more detailed error information.

See also: `setreuid` ([1021](#))

61.4.21 `setreuid`

Declaration: `function setreuid(ruid: uid_t; euid: uid_t) : cint`

Visibility: default

Description: `setreuid` sets the real user ID to `ruid` and the effective user ID to `euid`. Passing a value of -1 tells the system not to change that value.

Errors: On Error, -1 is returned. `fpgeterrno` can be used to get more detailed error information.

See also: `setregid` ([1021](#))

61.4.22 `statx`

Declaration: `function statx(dfd: cint; filename: PChar; flags: cuint; mask: cuint;
var buf: tstatx) : cint`

Visibility: default

61.4.23 sync_file_range

Synopsis: Force committing of data to disk.

Declaration: `function sync_file_range(fd: cint; offset: off64_t; nbytes: off64_t; flags: cuint) : cint`

Visibility: default

Description: `sync_file_range` forces the Linux kernel to write any data pages of a specified file (file descriptor `fd`) to disk. The range of the file is specified by the offset `offset` and the number of bytes `nbytes`. `Options` is an OR-ed combination of

SYNC_FILE_RANGE_WAIT_BEFORE Wait for write-out of previously-submitted specified pages before writing more data.

SYNC_FILE_RANGE_WRITE Initiate write of all dirty pages in the specified range.

SYNC_FILE_RANGE_WAIT_AFTER Wait upon write-out of specified pages in the range after performing any write.

If none is specified, the operation does nothing.

Errors: On return -1 is returned and `fperrno` is set to the actual error code. See the Linux man page for more on the error codes.

See also: `fdatsync` ([1018](#))

61.4.24 Sysinfo

Synopsis: Return kernel system information.

Declaration: `function Sysinfo(Info: PSysInfo) : cint`

Visibility: default

Description: `SysInfo` returns system information in `Info`. The function result is 0 if the call was successful. Returned information in `Info` includes:

uptime Number of seconds since boot.

loads 1, 5 and 15 minute load averages.

totalram total amount of main memory.

freeram amount of free memory.

sharedram amount of shared memory.

bufferram amount of memory used by buffers.

totalswap total amount of swap space.

freeswap amount of free swap space.

procs number of current processes.

Errors: On error, -1 is returned.

See also: `#rtl.baseunix.fpUname` ([243](#))

Listing: `./examples/linuxex/ex64.pp`

```

program Example64;

{$R-}
{ Example to demonstrate the SysInfo function.
  Sysinfo is Linux-only. }

{$ifdef Linux}
Uses Linux;

Function Mb(L : Longint) : longint;

begin
  Mb:=L div (1024*1024);
end;

Var Info : TSysInfo;
      D,M,Secs,H : longint;
{$endif}

begin
  {$ifdef Linux}
  If Not (SysInfo(@Info)=0) then
    Halt(1);
  With Info do
    begin
      D:=Uptime div (3600*24);
      UpTime:=UpTime mod (3600*24);
      h:=uptime div 3600;
      uptime:=uptime mod 3600;
      m:=uptime div 60;
      secs:=uptime mod 60;
      Writeln('Uptime : ',d,'days, ',h,' hours, ',m,' min, ',secs,' s. ');
      Writeln('Loads : ',Loads[1],',',Loads[2],',',Loads[3]);
      Writeln('Total Ram : ',Mb(totalram),'Mb. ');
      Writeln('Free Ram : ',Mb(freeram),'Mb. ');
      Writeln('Shared Ram : ',Mb(sharedram),'Mb. ');
      Writeln('Buffer Ram : ',Mb(bufferram),'Mb. ');
      Writeln('Total Swap : ',Mb(totalswap),'Mb. ');
      Writeln('Free Swap : ',Mb(freeswap),'Mb. ');
    end;
  {$endif}
end.

```

61.4.25 utimensat

Declaration: `function utimensat(dfd: cint; path: PChar;
const times: tkernel_timespecs; flags: cint) : cint`

Visibility: default

61.5 EPoll_Event

```

EPoll_Event = packed record
  Events : cuint32;

```



```

    Data : TEPoll_Data
;
end

```

Structure used in `epoll_ctl` ([1017](#)) call.

61.6 inotify_event

```

inotify_event = record
    wd : cint;
    mask : cuint32;
    cookie : cuint32
;
    len : cuint32;
    name : Char;
end

```

`inotify_event` is the structure used to report changes in a directory. When reading a `inotify` file descriptor, one or more `inotify_event` records can be read from the file descriptor.

61.7 kernel_timespec

```

kernel_timespec = record
    tv_sec : kernel_time64_t;
    tv_nsec : clonglong
;
end

```

61.8 statx_timestamp

```

statx_timestamp = record
    tv_sec : __s64;
    tv_nsec : __u32;
    __reserved
    : __s32;
end

```

61.9 tstatx

```

tstatx = record
    stx_mask : __u32;
    stx_blksize : __u32;
    stx_attributes
    : __u64;
    stx_nlink : __u32;
    stx_uid : __u32;

```

```

stx_gid : __u32
;
stx_mode : __u16;
__spare0 : Array[0..0] of __u16;
stx_ino
: __u64;
stx_size : __u64;
stx_blocks : __u64;
stx_attributes_mask
: __u64;
stx_atime : statx_timestamp;
stx_btime : statx_timestamp
;
stx_ctime : statx_timestamp;
stx_mtime : statx_timestamp;
stx_rdev_major : __u32;
stx_rdev_minor : __u32;
stx_dev_major
: __u32;
stx_dev_minor : __u32;
__spare2 : Array[0..13] of __u64
;
end

```

61.10 TSysInfo

```

TSysInfo = record
  uptime : clong;
  loads : Array[0..2] of culong
;
  totalram : culong;
  freeram : culong;
  sharedram : culong;
  bufferram : culong;
  totalswap : culong;
  freeswap : culong;
  procs : cushort;
  pad : cushort;
  totalhigh : culong;
  freehigh
  : culong;
  mem_unit : cuint;
end

```

Record with system information, used by the SysInfo ([1022](#)) call.

61.11 user_cap_data

```

user_cap_data = record
  effective : cuint32;

```

```
    permitted : cuint32
;
    inheritable : cuint32;
end
```

`user_cap_data` describes the set of capabilities for the indicated thread.

61.12 `user_cap_header`

```
user_cap_header = record
    version : cuint32;
    pid : cint;
end
```

`user_cap_header` describes the root user capabilities for the current thread, as set by `capget` (1014) and `capset` (1014)

61.13 `user_desc`

```
user_desc = record
    entry_number : cint;
    base_addr : cuint;
    limit : cuint;
    flags : cuint;
end
```

`user_desc` is the TLS (Thread Local Storage) segment descriptor used in the `Clone` call. It should not be used, as it contains highly kernel-specific data.

Chapter 62

Reference for unit 'Infodwrf'

62.1 Used units

Table 62.1: Used units by unit 'Infodwrf'

Name	Page
System	1335

62.2 Overview

The `Infodwrf` provides a routine that reads the debug information of an executable (if any exists) and returns source code information about this address. It works with DWARF debug information. Note that this unit is not thread-safe, and that its behaviour is undefined if multiple threads try to write a backtrace at the same time.

For stabs debug information, the `lineinfo` ([994](#)) unit must be used.

62.3 Constants, types and variables

62.3.1 Types

`CodePointer = Pointer`

`CodePointer` is added for 16-bit dos compatibility.

62.3.2 Variables

`AllowReuseOfLineInfoData : Boolean = True`

`AllowReuseOfLineInfoData` can be set to `True` to keep the last opened file open. When regularly creating backtraces (e.g. in a program log), this will significantly speed up operations.

62.4 Procedures and functions

62.4.1 CloseDwarf

Synopsis: Close DWARF info file descriptor.

Declaration: `procedure CloseDwarf`

Visibility: default

Description: `CloseDwarf` will close the file descriptor that was used to read DWARF debug information. This is useful if `AllowReuseOfLineInfoData` ([1027](#)) is used to cache DWARF information.

Errors: None.

See also: `AllowReuseOfLineInfoData` ([1027](#))

62.4.2 DwarfBackTraceStr

Synopsis: Get a backtrace from an address.

Declaration: `function DwarfBackTraceStr(addr: CodePointer) : string`

Visibility: default

Description: `DwarfBackTraceStr` returns a backtrace from a memory address `Addr`.

This is the actual callback for the backtrace handler `System.BackTraceStrFunc` ([1027](#)).

Errors: None.

See also: `GetLineInfo` ([1028](#))

62.4.3 GetLineInfo

Synopsis: Return source line information about an address.

Declaration: `function GetLineInfo(addr: CodePtrUInt; var func: string;
var source: string; var line: LongInt) : Boolean`

Visibility: default

Description: `GetLineInfo` returns source line information about the address `addr`. It searches this information in the DWARF debugging information found in the binary: If the file was compiled without debug information, nothing will be returned. Upon successful retrieval of the debug information, `True` is returned, and the `func` parameter is filled with the name of the function in which the address is located. The `source` parameter contains the name of the file in which the function was implemented, and `line` contains the line number in the source file for `addr`.

Errors: If no debug information is found, `False` is returned.

Chapter 63

Reference for unit 'Math'

63.1 Used units

Table 63.1: Used units by unit 'Math'

Name	Page
System	1335
sysutils	1609

63.2 Overview

This document describes the `math` unit. The `math` unit was initially written by Florian Klaempfl. It provides mathematical functions which aren't covered by the system unit.

This chapter starts out with a definition of all types and constants that are defined, after which an overview is presented of the available functions, grouped by category, and the last part contains a complete explanation of each function.

The following things must be taken into account when using this unit:

1. This unit is compiled in Object Pascal mode so all `integers` are 32 bit.
2. Some overloaded functions exist for data arrays of integers and floats. When using the address operator (`@`) to pass an array of data to such a function, make sure the address is typecasted to the right type, or turn on the 'typed address operator' feature. failing to do so, will cause the compiler not be able to decide which function you want to call.

63.3 Cash flow functions.

The cash flow functions in the `math` unit resolve the following equation:

$$FV + PV * q^n + PMT (q^n - 1) / (q - 1) = 0$$

In this formula, the following variables are present:

FV Future value

PV Present value

PMT Payment per period

n Number of payments (number of periods)

q> Interest Rate (return rate)

The financial functions FutureValue (1048), NumberOfPeriods (1064), Payment (1065), PresentValue (1068) and InterestRate (1051) solve this equation for one of the variables, when the other variables are known.

See also: FutureValue (1048), NumberOfPeriods (1064), Payment (1065), PresentValue (1068), TPaymentTime (1034)

63.4 Geometrical functions.

Table 63.2:

Name	Description
hypot (1050)	Hypotenuse of triangle
norm (1064)	Euclidean norm

63.5 Statistical functions.

Table 63.3:

Name	Description
mean (1058)	Mean of values
meanandstddev (1059)	Mean and standard deviation of values
momentskewkurtosis (1062)	Moments, skew and kurtosis
popnstddev (1065)	Population standard deviation
popnvariance (1066)	Population variance
randg (1070)	Gaussian distributed random value
stddev (1075)	Standard deviation
sum (1076)	Sum of values
sumofsquares (1077)	Sum of squared values
sumsandsquares (1077)	Sum of values and squared values
totalvariance (1080)	Total variance of values
variance (1080)	variance of values

63.6 Number converting.

Table 63.4:

Name	Description
<code>ceil</code> (1040)	Round to infinity
<code>floor</code> (1046)	Round to minus infinity
<code>frexp</code> (1047)	Return mantissa and exponent

63.7 Exponential and logarithmic functions.

Table 63.5:

Name	Description
<code>intpower</code> (1051)	Raise float to integer power
<code>ldexp</code> (1053)	Calculate $2^x \times \text{float}$
<code>lnxp1</code> (1053)	calculate $\log(x+1)$
<code>log10</code> (1054)	calculate 10-base log
<code>log2</code> (1054)	calculate 2-base log
<code>logn</code> (1055)	calculate N-base log
<code>power</code> (1067)	raise float to arbitrary power

63.8 Hyperbolic functions.

Table 63.6:

Name	Description
<code>arcosh</code> (1036)	calculate reverse hyperbolic cosine
<code>arsinh</code> (1039)	calculate reverse hyperbolic sine
<code>artanh</code> (1039)	calculate reverse hyperbolic tangent
<code>cosh</code> (1041)	calculate hyperbolic cosine
<code>sinh</code> (1074)	calculate hyperbolic sine
<code>tanh</code> (1079)	calculate hyperbolic tangent

63.9 Trigonometric functions.

Table 63.7:

Name	Description
<code>arccos</code> (1035)	calculate reverse cosine
<code>arcsin</code> (1037)	calculate reverse sine
<code>arctan2</code> (1038)	calculate reverse tangent
<code>cotan</code> (1042)	calculate cotangent
<code>sincos</code> (1074)	calculate sine and cosine
<code>tan</code> (1078)	calculate tangent

63.10 Angle unit conversion.

Routines to convert angles between different angle units.

Table 63.8:

Name	Description
<code>cycletorad</code> (1043)	convert cycles to radians
<code>degtograd</code> (1044)	convert degrees to grads
<code>degtorad</code> (1045)	convert degrees to radians
<code>gradtodeg</code> (1049)	convert grads to degrees
<code>gradtorad</code> (1049)	convert grads to radians
<code>radto cycle</code> (1068)	convert radians to cycles
<code>radtodeg</code> (1069)	convert radians to degrees
<code>radto grad</code> (1069)	convert radians to grads

63.11 Min/max determination.

Functions to determine the minimum or maximum of numbers:

Table 63.9:

Name	Description
<code>max</code> (1056)	Maximum of 2 values
<code>maxIntValue</code> (1056)	Maximum of an array of integer values
<code>maxvalue</code> (1057)	Maximum of an array of values
<code>min</code> (1060)	Minimum of 2 values
<code>minIntValue</code> (1060)	Minimum of an array of integer values
<code>minvalue</code> (1061)	Minimum of an array of values

63.12 Constants, types and variables

63.12.1 Constants

`EqualsValue = 0`

Values are the same.

`GreaterThanValue = High(TValueRelationship)`

First values is greater than second value.

`Infinity = 1.0 / 0.0`

Value is infinity.

`LessThanValue = Low(TValueRelationship)`

First value is less than second value.

`MaxFloat = 0`

Maximum value of float type.

`MinFloat = 0`

Minimum value (closest to zero) of float type.

`NaN = 0.0 / 0.0`

Value is Not a Number.

`NegativeValue = Low(TValueSign)`

Value is negative.

`NegInfinity = (- 1.0) / (0.0)`

Value is negative (minus) infinity.

`PositiveValue = High(TValueSign)`

Value is positive.

`ZeroValue = 0`

Value is zero.

63.12.2 Types

`Float = MaxFloatType`

All calculations are done with the `Float` type which is the largest float type available for the current CPU. This allows to recompile the unit with a different float type to obtain a desired precision. The pointer type `PFloat` ([1034](#)) is used in functions that accept an array of values of arbitrary length.

`PFloat = ^Float`

Pointer to `Float` ([1034](#)) type.

`PInteger = ObjPas.PInteger`

Pointer to integer type.

`TFPUException = system.TFPUException`

Type describing Floating Point processor exceptions.

`TFPUExceptionMask = system.TFPUExceptionMask`

Type to set the Floating Point Unit exception mask.

`TFPUPrecisionMode = system.TFPUPrecisionMode`

Type describing the default precision for the Floating Point processor.

`TFPURoundingMode = system.TFPURoundingMode`

Type describing the rounding mode for the Floating Point processor.

`TPaymentTime = (ptEndOfPeriod, ptStartOfPeriod)`

Table 63.10: Enumeration values for type `TPaymentTime`

Value	Explanation
<code>ptEndOfPeriod</code>	End of period.
<code>ptStartOfPeriod</code>	Start of period.

Type used in financial (interest) calculations.

`TRoundToRange = - 37..37`

`TRoundToRange` is the range of valid digits to be used in the `RoundTo` ([1071](#)) function.

`TValueRelationship = - 1..1`

Type to describe relational order between values.

`TValueSign = - 1..1`

Type indicating sign of a value.

63.13 Procedures and functions

63.13.1 ArcCos

Synopsis: Return inverse cosine.

Declaration: `function ArcCos(x: Single) : Single`
`function ArcCos(x: Double) : Double`
`function ArcCos(x: Extended) : Extended`

Visibility: default

Description: `Arccos` returns the inverse cosine of its argument `x`. The argument `x` should lie between -1 and 1 (borders included).

Errors: If the argument `x` is not in the allowed range, an `EInvalidArgument` exception is raised.

See also: `arcsin` ([1037](#)), `arcosh` ([1036](#)), `arsinh` ([1039](#)), `artanh` ([1039](#))

Listing: `./examples/mathex/ex1.pp`

Program Example1;

{ Program to demonstrate the arccos function. }

Uses math;

Procedure WriteRadDeg(X : float);

begin

`Writeln(X:8:5, ' rad = ', radtodeg(x):8:5, ' degrees.')`

end;

begin

`WriteRadDeg (arccos(1));`

`WriteRadDeg (arccos(sqrt(3)/2));`

`WriteRadDeg (arccos(sqrt(2)/2));`

`WriteRadDeg (arccos(1/2));`

`WriteRadDeg (arccos(0));`

`WriteRadDeg (arccos(-1));`

end.

63.13.2 ArcCosH

Synopsis: Return inverse hyperbolic cosine.

Declaration: `function ArcCosH(x: Float) : Float`

Visibility: default

Description: `arccosh` returns the inverse hyperbolic cosine of its argument `x`.

This function is an alias for `arcosh` ([1036](#)), provided for Delphi compatibility.

See also: `arcosh` ([1036](#))

63.13.3 ArcCot

Declaration: `function ArcCot(X: Single) : Single`
`function ArcCot(X: Double) : Double`
`function ArcCot(X: Extended) : Extended`

Visibility: default

63.13.4 ArcCotH

Declaration: `function ArcCotH(X: Single) : Single`
`function ArcCotH(X: Double) : Double`
`function ArcCotH(X: Extended) : Extended`

Visibility: default

63.13.5 ArcCsc

Declaration: `function ArcCsc(X: Single) : Single`
`function ArcCsc(X: Double) : Double`
`function ArcCsc(X: Extended) : Extended`

Visibility: default

63.13.6 ArcCscH

Declaration: `function ArcCscH(X: Single) : Single`
`function ArcCscH(X: Double) : Double`
`function ArcCscH(X: Extended) : Extended`

Visibility: default

63.13.7 ArCosh

Synopsis: Return inverse hyperbolic cosine.

Declaration: `function ArCosh(x: Float) : Float`

Visibility: default

Description: `ArCosh` returns the inverse hyperbolic cosine of its argument `x`. The argument `x` should be larger than 1. The `arccosh` variant of this function is supplied for Delphi compatibility.

Errors: If the argument `x` is not in the allowed range, an `EInvalidArgument` exception is raised.

See also: `cosh` ([1041](#)), `sinh` ([1074](#)), `arcsin` ([1037](#)), `arsinh` ([1039](#)), `artanh` ([1039](#)), `tanh` ([1079](#))

Listing: `./examples/mathex/ex3.pp`

Program Example3;

{ Program to demonstrate the arcosh function. }

Uses math;

begin

```

Writeln(arcosh(1));
Writeln(arcosh(2));
end.

```

63.13.8 ArcSec

```

Declaration: function ArcSec(X: Single) : Single
             function ArcSec(X: Double) : Double
             function ArcSec(X: Extended) : Extended

```

Visibility: default

63.13.9 ArcSecH

```

Declaration: function ArcSecH(X: Single) : Single
             function ArcSecH(X: Double) : Double
             function ArcSecH(X: Extended) : Extended

```

Visibility: default

63.13.10 ArcSin

Synopsis: Return inverse sine.

```

Declaration: function ArcSin(x: Single) : Single
             function ArcSin(x: Double) : Double
             function ArcSin(x: Extended) : Extended

```

Visibility: default

Description: `Arcsin` returns the inverse sine of its argument `x`. The argument `x` should lie between -1 and 1.

Errors: If the argument `x` is not in the allowed range, an `EInvalidArgument` exception is raised.

See also: `arccos` ([1035](#)), `arcosh` ([1036](#)), `arsinh` ([1039](#)), `artanh` ([1039](#))

Listing: `./examples/mathex/ex2.pp`

Program Example1;

{ Program to demonstrate the arcsin function. }

Uses math;

Procedure WriteRadDeg(X : float);

begin

Writeln(X:8:5, ' rad = ', radtodeg(x):8:5, ' degrees.')

end;

begin

 WriteRadDeg (arcsin(1));

 WriteRadDeg (arcsin(**sqrt**(3)/2));

 WriteRadDeg (arcsin(**sqrt**(2)/2));

 WriteRadDeg (arcsin(1/2));

 WriteRadDeg (arcsin(0));

```
WriteRadDeg (arcsin(-1));
end.
```

63.13.11 ArcSinH

Synopsis: Return inverse hyperbolic sine.

Declaration: `function ArcSinH(x: Float) : Float`

Visibility: default

Description: `arsinh` returns the inverse hyperbolic sine of its argument `x`.

This function is an alias for `arsinh` ([1039](#)), provided for Delphi compatibility.

See also: `arsinh` ([1039](#))

63.13.12 ArcTan2

Synopsis: Return arctangent of (y/x).

Declaration: `function ArcTan2(y: Float; x: Float) : Float`

Visibility: default

Description: `arctan2` calculates `arctan(y/x)`, and returns an angle in the correct quadrant. The returned angle will be in the range $-\pi$ to π radians. The values of `x` and `y` must be between -2^{64} and 2^{64} , moreover `x` should be different from zero. On Intel systems this function is implemented with the native intel `fpatan` instruction.

See also: `arccos` ([1035](#)), `arcosh` ([1036](#)), `arsinh` ([1039](#)), `artanh` ([1039](#))

Listing: `./examples/mathex/ex6.pp`

Program Example6;

{ Program to demonstrate the arctan2 function. }

Uses math;

Procedure WriteRadDeg(X : float);

begin

WriteLn(X:8:5, ' rad = ', radtodeg(x):8:5, ' degrees.')

end;

begin

WriteRadDeg (arctan2(2,1));

end.

63.13.13 ArcTanH

Synopsis: Return inverse hyperbolic tangent.

Declaration: `function ArcTanH(x: Float) : Float`

Visibility: default

Description: `arsinh` returns the inverse hyperbolic tangent of its argument `x`.

This function is an alias for `artanh` (1039), provided for Delphi compatibility.

See also: `artanh` (1039)

63.13.14 ArSinH

Synopsis: Return inverse hyperbolic sine.

Declaration: `function ArSinH(x: Float) : Float`

Visibility: default

Description: `arsinh` returns the inverse hyperbolic sine of its argument `x`. The `arscsinh` variant of this function is supplied for Delphi compatibility.

Errors: None.

See also: `arcosh` (1036), `arccos` (1035), `arcsin` (1037), `artanh` (1039)

Listing: `./examples/mathex/ex4.pp`

Program Example4;

{ Program to demonstrate the arsinh function. }

Uses math;

begin

Writeln(arsinh(0));

Writeln(arsinh(1));

end.

63.13.15 ArTanH

Synopsis: Return inverse hyperbolic tangent.

Declaration: `function ArTanH(x: Float) : Float`

Visibility: default

Description: `artanh` returns the inverse hyperbolic tangent of its argument `x`, where `x` should lie in the interval `[-1,1]`, borders included. The `arctanh` variant of this function is supplied for Delphi compatibility.

Errors: In case `x` is not in the interval `[-1,1]`, an `EInvalidArgument` exception is raised.

See also: `arcosh` (1036), `arccos` (1035), `arcsin` (1037), `artanh` (1039)

Listing: `./examples/mathex/ex5.pp`

Program Example5;

{ Program to demonstrate the artanh function. }

Uses math;

begin

Writeln(artanh(0));

Writeln(artanh(0.5));

end.

63.13.16 Ceil

Synopsis: Return the lowest integer number greater than or equal to argument.

Declaration: `function Ceil(x: Float) : Integer`

Visibility: default

Description: `Ceil` returns the lowest integer number greater than or equal to `x`. The absolute value of `x` should be less than `maxint`.

Errors: If the absolute value of `x` is larger than `maxint`, an overflow error will occur.

See also: `floor` ([1046](#))

Listing: `./examples/mathex/ex7.pp`

Program Example7;

{ Program to demonstrate the Ceil function. }

Uses math;

begin

Writeln(Ceil(-3.7)); *// should be -3*

Writeln(Ceil(3.7)); *// should be 4*

Writeln(Ceil(-4.0)); *// should be -4*

end.

63.13.17 Ceil64

Synopsis: Round to the nearest bigger int64 value.

Declaration: `function Ceil64(x: Float) : Int64`

Visibility: default

Description: `Ceil64` rounds the value `X` to the nearest bigger int64 value. The result is always bigger than `X`.

Errors: None.

See also: `ceil` ([1040](#)), `floor64` ([1047](#))

63.13.18 ClearExceptions

Synopsis: Clear Floating Point Unit exceptions.

Declaration: `procedure ClearExceptions(RaisePending: Boolean)`

Visibility: default

Description: Clear Floating Point Unit exceptions.

63.13.19 CompareValue

Synopsis: Compare 2 values.

Declaration: `function CompareValue(const A: Integer; const B: Integer) : TValueRelationship`
`function CompareValue(const A: Int64; const B: Int64) : TValueRelationship`
`function CompareValue(const A: QWord; const B: QWord) : TValueRelationship`
`function CompareValue(const A: Single; const B: Single; delta: Single) : TValueRelationship`
`function CompareValue(const A: Double; const B: Double; delta: Double) : TValueRelationship`
`function CompareValue(const A: Extended; const B: Extended; delta: Extended) : TValueRelationship`

Visibility: default

Description: CompareValue compares 2 integer or floating point values A and B and returns one of the following values:

-1 if A<B
0 if A=B
1 if A>B

See also: TValueRelationship ([1034](#))

63.13.20 Cosecant

Synopsis: Calculate cosecant.

Declaration: `function Cosecant(x: Float) : Float`

Visibility: default

Description: cosecant calculates the cosecant ($1/\sin(x)$) of its argument x.

Errors: If 0 or 180 degrees is specified, an exception will be raised.

See also: secant ([1072](#))

63.13.21 cosh

Synopsis: Return hyperbolic cosine.

Declaration: `function cosh(x: Single) : Single`
`function cosh(x: Double) : Double`
`function cosh(x: Extended) : Extended`

Visibility: default

Description: Cosh returns the hyperbolic cosine of its argument {x}.

Errors: None.

See also: arcosh ([1036](#)), sinh ([1074](#)), arsinh ([1039](#))

Listing: ./examples/mathex/ex8.pp

Program Example8;

{ Program to demonstrate the cosh function. }

Uses math;

begin

Writeln(Cosh(0));

Writeln(Cosh(1));

end.

63.13.22 Cot

Synopsis: Alias for `Cotan`.

Declaration: `function Cot(x: Float) : Float`

Visibility: default

Description: `cot` is an alias for the `cotan` ([1042](#)) function.

See also: `cotan` ([1042](#))

63.13.23 Cotan

Synopsis: Return cotangent.

Declaration: `function Cotan(x: Float) : Float`

Visibility: default

Description: `Cotan` returns the cotangent of its argument `x`. The argument `x` must be in radians. `x` should be different from zero.

Errors: If `x` is zero then a overflow error will occur.

See also: `tanh` ([1079](#))

Listing: ./examples/mathex/ex9.pp

Program Example9;

{ Program to demonstrate the cotan function. }

Uses math;

begin

writeln(cotan(**pi**/2));

Writeln(cotan(**pi**/3));

Writeln(cotan(**pi**/4));

end.

63.13.24 CotH

Declaration: `function CotH(const X: Single) : Single`
`function CotH(const X: Double) : Double`
`function CotH(const X: Extended) : Extended`

Visibility: default

63.13.25 Csc

Synopsis: Alias for cosecant.

Declaration: `function Csc(x: Float) : Float`

Visibility: default

Description: `csc` is an alias for the cosecant (1041) function.

See also: cosecant (1041)

63.13.26 CscH

Declaration: `function CscH(const X: Single) : Single`
`function CscH(const X: Double) : Double`
`function CscH(const X: Extended) : Extended`

Visibility: default

63.13.27 CycleToDeg

Declaration: `function CycleToDeg(const Cycles: Single) : Single`
`function CycleToDeg(const Cycles: Double) : Double`
`function CycleToDeg(const Cycles: Extended) : Extended`

Visibility: default

63.13.28 CycleToGrad

Declaration: `function CycleToGrad(const Cycles: Single) : Single`
`function CycleToGrad(const Cycles: Double) : Double`
`function CycleToGrad(const Cycles: Extended) : Extended`

Visibility: default

63.13.29 CycleToRad

Synopsis: Convert cycle angle to radians angle.

Declaration: `function CycleToRad(const Cycles: Single) : Single`
`function CycleToRad(const Cycles: Double) : Double`
`function CycleToRad(const Cycles: Extended) : Extended`

Visibility: default

Description: `CycleToRad` transforms it's argument `cycle` (an angle expressed in cycles) to radians. (1 cycle is 2π radians).

Errors: None.

See also: [degtograd \(1044\)](#), [degtorad \(1045\)](#), [radtoDeg \(1069\)](#), [radtoGrad \(1069\)](#), [radtoCycle \(1068\)](#)

Listing: ./examples/mathex/ex10.pp

Program Example10;

{ Program to demonstrate the cycletorad function. }

Uses math;

begin

writeln(cos(cycletorad(1/6))); *// Should print 1/2*

writeln(cos(cycletorad(1/8))); *// should be sqrt(2)/2*

end.

63.13.30 DegNormalize

Synopsis: Normalize an angle measured in degrees.

Declaration: function DegNormalize(deg: single) : single
 function DegNormalize(deg: Double) : Double
 function DegNormalize(deg: extended) : extended

Visibility: default

Description: DegNormalize returns the angle deg as a positive angle between 0 and 360 degrees.

63.13.31 DegToCycle

Declaration: function DegToCycle(const Degrees: Single) : Single
 function DegToCycle(const Degrees: Double) : Double
 function DegToCycle(const Degrees: Extended) : Extended

Visibility: default

63.13.32 DegToGrad

Synopsis: Convert degree angle to grads angle.

Declaration: function DegToGrad(deg: Float) : Float

Visibility: default

Description: Degtograd transforms it's argument deg (an angle in degrees) to grads. (90 degrees is 100 grad.)

Errors: None.

See also: [cycletorad \(1043\)](#), [degtorad \(1045\)](#), [radtoDeg \(1069\)](#), [radtoGrad \(1069\)](#), [radtoCycle \(1068\)](#)

Listing: ./examples/mathex/ex11.pp

Program Example11;

{ Program to demonstrate the degtograd function. }

Uses math;

begin

writeln(deltograd(90));
 writeln(deltograd(180));
 writeln(deltograd(270))

end.

63.13.33 DegToRad

Synopsis: Convert degree angle to radians angle.

Declaration: `function DegToRad(deg: Float) : Float`

Visibility: default

Description: Deltograd converts it's argument deg (an angle in degrees) to radians. (pi radians is 180 degrees)

Errors: None.

See also: cycletorad ([1043](#)), degtograd ([1044](#)), radto deg ([1069](#)), radtograd ([1069](#)), radto cycle ([1068](#))

Listing: ./examples/mathex/ex12.pp

Program Example12;

{ Program to demonstrate the degtorad function. }

Uses math;

begin

writeln(deltorad(45));
 writeln(deltorad(90));
 writeln(deltorad(180));
 writeln(deltorad(270));
 writeln(deltorad(360));

end.

63.13.34 DivMod

Synopsis: Return DIV and MOD of arguments.

Declaration: `procedure DivMod(Dividend: LongInt; Divisor: Word; var Result: Word;
 var Remainder: Word)
 procedure DivMod(Dividend: LongInt; Divisor: Word;
 var Result: SmallInt; var Remainder: SmallInt)
 procedure DivMod(Dividend: DWord; Divisor: DWord; var Result: DWord;
 var Remainder: DWord)
 procedure DivMod(Dividend: LongInt; Divisor: LongInt;
 var Result: LongInt; var Remainder: LongInt)`

Visibility: default

Description: `DivMod` returns Dividend **DIV** Divisor in Result, and Dividend **MOD** Divisor in Remainder

63.13.35 EnsureRange

Synopsis: Change value so it fits in a specified range.

Declaration: `function EnsureRange(const AValue: Integer; const AMin: Integer; const AMax: Integer) : Integer; Overload`
`function EnsureRange(const AValue: Int64; const AMin: Int64; const AMax: Int64) : Int64; Overload`
`function EnsureRange(const AValue: Double; const AMin: Double; const AMax: Double) : Double; Overload`

Visibility: default

Description: `EnsureRange` returns Value if AValue is in the range AMin..AMax. It returns AMin if the value is less than AMin, or AMax if the value is larger than AMax.

See also: `InRange` ([1051](#))

63.13.36 ExpM1

Declaration: `function ExpM1(x: Double) : Double`
`function ExpM1(x: extended) : extended`

Visibility: default

63.13.37 Floor

Synopsis: Return the largest integer smaller than or equal to argument.

Declaration: `function Floor(x: Float) : Integer`

Visibility: default

Description: `Floor` returns the largest integer smaller than or equal to x. The absolute value of x should be less than `maxint`.

Errors: If x is larger than `maxint`, an overflow will occur.

See also: `ceil` ([1040](#))

Listing: `./examples/mathex/ex13.pp`

Program Example13;

{ Program to demonstrate the floor function. }

Uses math;

begin

Writeln(Floor(-3.7)); *// should be -4*

Writeln(Floor(3.7)); *// should be 3*

Writeln(Floor(-4.0)); *// should be -4*

end.

63.13.38 Floor64

Synopsis: Round to the nearest smaller int64 value.

Declaration: `function Floor64(x: Float) : Int64`

Visibility: default

Description: `Floor64` rounds the value `X` to the nearest smaller int64 value. The result is always smaller than `X`.

Errors: None.

See also: `floor` ([1046](#)), `ceil64` ([1040](#))

63.13.39 FMod

Synopsis: Floating point modulo.

Declaration: `function FMod(const a: Single; const b: Single) : Single; Overload`
`function FMod(const a: Double; const b: Double) : Double; Overload`
`function FMod(const a: Extended; const b: Extended) : Extended`
`; Overload`

Visibility: default

Description: `FMod` is the floating-point equivalent of the modulo operation `a mod b`. It returns the result of
`a - b * Int(a/b)`

Errors: `b` may not be zero, but no check is performed.

63.13.40 Frexp

Synopsis: Return mantissa and exponent.

Declaration: `procedure Frexp(X: single; out Mantissa: single; out Exponent: Integer)`
`procedure Frexp(X: Double; out Mantissa: Double; out Exponent: Integer)`
`procedure Frexp(X: extended; out Mantissa: extended;`
`out Exponent: Integer)`

Visibility: default

Description: `Frexp` returns the mantissa and exponent of its argument `x` in mantissa and exponent.

Errors: None

Listing: `./examples/mathex/ex14.pp`

Program Example14;

{ Program to demonstrate the frexp function. }

Uses math;

Procedure dofrep(**Const** X : extended);

var man : extended;
exp: longint;

begin


```

man:=0;
exp:=0;
frexp(x,man,exp);
write(x,' has ');
Writeln('mantissa ',man,' and exponent ',exp);
end;

```

```

begin
// dofrexp(1.00);
  dofrexp(1.02e-1);
  dofrexp(1.03e-2);
  dofrexp(1.02e1);
  dofrexp(1.03e2);
end.

```

63.13.41 FutureValue

Synopsis: Calculate the future value of an investment.

Declaration: `function FutureValue (ARate: Float; NPeriods: Integer; APayment: Float; APresentValue: Float; APaymentTime: TPaymentTime) : Float`

Visibility: default

Description: `FutureValue` calculates the future value of an investment (FV) in the cash flow formula (see `CashFlowFunctions` (1029)) The function result is the future value of an investment of `APresentValue` (PV), where `APayment` (PMT) is invested for `NPeriods` (n) at the rate of `ARate` (q) per period.

The `APaymentTime` parameter determines whether the investment (payment) is an ordinary annuity or an annuity due: `ptEndOfPeriod` must be used if payments are at the end of each period. If the payments are at the beginning of the periode, `ptStartOfPeriod` must be used.

See also: `InterestRate` (1051), `NumberOfPeriods` (1064), `Payment` (1065), `PresentValue` (1068), `TPaymentTime` (1034), `CashFlowFunctions` (1029)

63.13.42 GetExceptionMask

Synopsis: Get the Floating Point Unit exception mask.

Declaration: `function GetExceptionMask : TFPUExceptionMask`

Visibility: default

Description: Get the Floating Point Unit exception mask.

63.13.43 GetPrecisionMode

Synopsis: Return the Floating Point Unit precision mode.

Declaration: `function GetPrecisionMode : TFPUPrecisionMode`

Visibility: default

Description: Return the Floating Point Unit precision mode.

63.13.44 GetRoundMode

Synopsis: Return the Floating Point Unit rounding mode.

Declaration: `function GetRoundMode : TFPURoundingMode`

Visibility: default

Description: Return the Floating Point Unit rounding mode.

63.13.45 GradToCycle

Declaration: `function GradToCycle(const Grads: Single) : Single`
`function GradToCycle(const Grads: Double) : Double`
`function GradToCycle(const Grads: Extended) : Extended`

Visibility: default

63.13.46 GradToDeg

Synopsis: Convert grads angle to degrees angle.

Declaration: `function GradToDeg(grad: Float) : Float`

Visibility: default

Description: `Gradtodeg` converts its argument `grad` (an angle in grads) to degrees. (100 grad is 90 degrees)

Errors: None.

See also: `cycletorad` ([1043](#)), `degtograd` ([1044](#)), `radtodeg` ([1069](#)), `radtograd` ([1069](#)), `radtocycle` ([1068](#)), `gradtorad` ([1049](#))

Listing: `./examples/mathex/ex15.pp`

Program Example15;

{ Program to demonstrate the gradtodeg function. }

Uses math;

begin
`writeln(gradtodeg(100));`
`writeln(gradtodeg(200));`
`writeln(gradtodeg(300));`
end.

63.13.47 GradToRad

Synopsis: Convert grads angle to radians angle.

Declaration: `function GradToRad(grad: Float) : Float`

Visibility: default

Description: `Gradtorad` converts its argument `grad` (an angle in grads) to radians. (200 grad is pi degrees).

Errors: None.

See also: [cycleto rad \(1043\)](#), [degtograd \(1044\)](#), [radto deg \(1069\)](#), [radto grad \(1069\)](#), [radto cycle \(1068\)](#), [gradto deg \(1049\)](#)

Listing: ./examples/mathex/ex16.pp

Program Example16;

{ Program to demonstrate the gradtorad function. }

Uses math;

begin

writeln(gradtorad(100));

writeln(gradtorad(200));

writeln(gradtorad(300));

end.

63.13.48 Hypot

Synopsis: Return hypotenuse of triangle.

Declaration: `function Hypot(x: Float; y: Float) : Float`

Visibility: default

Description: `Hypot` returns the hypotenuse of the triangle where the sides adjacent to the square angle have lengths `x` and `y`. The function uses Pythagoras' rule for this.

Errors: None.

Listing: ./examples/mathex/ex17.pp

Program Example17;

{ Program to demonstrate the hypot function. }

Uses math;

begin

Writeln(hypot(3,4)); *// should be 5*

end.

63.13.49 IfThen

Synopsis: Return one of two values, depending on a boolean condition.

Declaration: `function IfThen(val: Boolean; const iftrue: Integer;
 const iffalse: Integer) : Integer; Overload`
`function IfThen(val: Boolean; const iftrue: Int64; const iffalse: Int64)
 : Int64; Overload`
`function IfThen(val: Boolean; const iftrue: Double;
 const iffalse: Double) : Double; Overload`

Visibility: default

Description: `ifthen` returns `iftrue` if `val` is `True`, and `iffalse` if `val` is `False`.

This function can be used in expressions.

63.13.50 InRange

Synopsis: Check whether value is in range.

Declaration:

```
function InRange(const AValue: Integer; const AMin: Integer;
                 const AMax: Integer) : Boolean; Overload
function InRange(const AValue: Int64; const AMin: Int64;
                 const AMax: Int64) : Boolean; Overload
function InRange(const AValue: Double; const AMin: Double;
                 const AMax: Double) : Boolean; Overload
```

Visibility: default

Description: `InRange` returns `True` if `AValue` is in the range `AMin..AMax`. It returns `False` if `Value` lies outside the specified range.

See also: `EnsureRange` ([1046](#))

63.13.51 InterestRate

Synopsis: Calculate the interest rate value of an investment.

Declaration:

```
function InterestRate(NPeriods: Integer; APayment: Float;
                     APresentValue: Float; AFutureValue: Float;
                     APaymentTime: TPaymentTime) : Float
```

Visibility: default

Description: `InterestRate` calculates the future value of an investment (q) in the cash flow formula (see [CashFlowFunctions](#) ([1029](#))). The function result is the interest rate value in case of a future value `AFutureValue` for an investment of a start value `APresentValue` (PV), where `APayment` (PMT) is invested for `NPeriods` (n).

The `APaymentTime` parameter determines whether the investment (payment) is an ordinary annuity or an annuity due: `ptEndOfPeriod` must be used if payments are at the end of each period. If the payments are at the beginning of the periode, `ptStartOfPeriod` must be used.

See also: `FutureValue` ([1048](#)), `NumberOfPeriods` ([1064](#)), `Payment` ([1065](#)), `PresentValue` ([1068](#)), `TPaymentTime` ([1034](#)), [CashFlowFunctions](#) ([1029](#))

63.13.52 IntPower

Synopsis: Return integer power.

Declaration:

```
function IntPower(base: Float; exponent: LongInt) : Float
```

Visibility: default

Description: `Intpower` returns `base` to the power `exponent`, where `exponent` is an integer value.

Errors: If `base` is zero and the `exponent` is negative, then an overflow error will occur.

See also: `power` ([1067](#))

Listing: `./examples/mathex/ex18.pp`

Program Example18;*{ Program to demonstrate the intpower function. }***Uses** math;**Procedure** DoIntpower (X : extended; Pow : Integer);**begin** **writeln**(X:8:4,'^',Pow:2,' = ',intpower(X,pow):8:4);**end;****begin**

dointpower(0.0,0);

dointpower(1.0,0);

dointpower(2.0,5);

dointpower(4.0,3);

dointpower(2.0,-1);

dointpower(2.0,-2);

dointpower(-2.0,4);

dointpower(-4.0,3);

end.

63.13.53 IsInfinite

Synopsis: Check whether value is infinite.

Declaration: function IsInfinite(const d: Single) : Boolean; Overload
 function IsInfinite(const d: Double) : Boolean; Overload
 function IsInfinite(const d: Extended) : Boolean; Overload

Visibility: default

Description: IsInfinite returns True if the double d contains the infinite value.

See also: IsZero ([1052](#)), IsInfinite ([1052](#))**63.13.54 IsNan**

Synopsis: Check whether value is Not a Number.

Declaration: function IsNan(const d: Single) : Boolean; Overload
 function IsNan(const d: Double) : Boolean; Overload
 function IsNan(const d: Extended) : Boolean; Overload

Visibility: default

Description: IsNan returns True if the double d contains Not A Number (a value which cannot be represented correctly in double format).

See also: IsZero ([1052](#)), IsInfinite ([1052](#))**63.13.55 IsZero**

Synopsis: Check whether value is zero.

Errors: If $x \leq -1$ then an `EInvalidArgument` exception will be raised.

See also: `ldexp` (1053), `log10` (1054), `log2` (1054), `logn` (1055)

Listing: ./examples/mathex/ex20.pp

Program Example20;

{ Program to demonstrate the lnxp1 function. }

Uses math;

begin

writeln(lnxp1(0));

writeln(lnxp1(0.5));

writeln(lnxp1(1));

end.

63.13.58 Log10

Synopsis: Return 10-Based logarithm.

Declaration: `function Log10(x: Float) : Float`

Visibility: default

Description: `Log10` returns the 10-base logarithm of `x`.

Errors: If `x` is less than or equal to 0 an 'invalid fpu operation' error will occur.

See also: `ldexp` (1053), `lnxp1` (1053), `log2` (1054), `logn` (1055)

Listing: ./examples/mathex/ex21.pp

Program Example21;

{ Program to demonstrate the log10 function. }

Uses math;

begin

Writeln(Log10(10):8:4);

Writeln(Log10(100):8:4);

Writeln(Log10(1000):8:4);

Writeln(Log10(1):8:4);

Writeln(Log10(0.1):8:4);

Writeln(Log10(0.01):8:4);

Writeln(Log10(0.001):8:4);

end.

63.13.59 Log2

Synopsis: Return 2-based logarithm.

Declaration: `function Log2(x: Float) : Float`

Visibility: default

Description: `Log2` returns the 2-base logarithm of `X`.

Errors: If `x` is less than or equal to 0 an 'invalid fpu operation' error will occur.

See also: `ldexp` ([1053](#)), `lnxp1` ([1053](#)), `log10` ([1054](#)), `logn` ([1055](#))

Listing: `./examples/mathex/ex22.pp`

Program `Example22`;

{ Program to demonstrate the log2 function. }

Uses `math`;

begin

Writeln(`Log2(2)`:8:4);

Writeln(`Log2(4)`:8:4);

Writeln(`Log2(8)`:8:4);

Writeln(`Log2(1)`:8:4);

Writeln(`Log2(0.5)`:8:4);

Writeln(`Log2(0.25)`:8:4);

Writeln(`Log2(0.125)`:8:4);

end.

63.13.60 LogN

Synopsis: Return N-based logarithm.

Declaration: `function LogN(n: Float; x: Float) : Float`

Visibility: `default`

Description: `Logn` returns the n-base logarithm of `X`.

Errors: If `x` is less than or equal to 0 an 'invalid fpu operation' error will occur.

See also: `ldexp` ([1053](#)), `lnxp1` ([1053](#)), `log10` ([1054](#)), `log2` ([1054](#))

Listing: `./examples/mathex/ex23.pp`

Program `Example23`;

{ Program to demonstrate the logn function. }

Uses `math`;

begin

Writeln(`Logn(3,4)`:8:4);

Writeln(`Logn(2,4)`:8:4);

Writeln(`Logn(6,9)`:8:4);

Writeln(`Logn(exp(1),exp(1))`:8:4);

Writeln(`Logn(0.5,1)`:8:4);

Writeln(`Logn(0.25,3)`:8:4);

Writeln(`Logn(0.125,5)`:8:4);

end.

63.13.61 Max

Synopsis: Returns the largest of two values.

Declaration: `function Max(a: Integer; b: Integer) : Integer; Overload`
`function Max(a: Int64; b: Int64) : Int64; Overload`
`function Max(a: QWord; b: QWord) : QWord; Overload`
`function Max(a: Single; b: Single) : Single; Overload`
`function Max(a: Double; b: Double) : Double; Overload`
`function Max(a: Extended; b: Extended) : Extended; Overload`

Visibility: default

Description: `Max` returns the largest of the two values in the `a` and `b` arguments. The overloaded routines use the same type for both of the compared arguments. The type for the first argument determines the overloaded function used for the comparison.

Errors: None.

See also: `min` ([1060](#)), `maxIntValue` ([1056](#)), `maxvalue` ([1057](#))

Listing: `./examples/mathex/ex24.pp`

Program Example24;

{ Program to demonstrate the max function. }

Uses math;

Var

A,B : Cardinal;

begin

A:=1;b:=2;

writeln(max(a,b));

end.

63.13.62 MaxIntValue

Synopsis: Return largest element in integer array.

Declaration: `function MaxIntValue(const Data: Array of Integer) : Integer`

Visibility: default

Description: `MaxIntValue` returns the largest integer out of the `Data` array.

This function is provided for Delphi compatibility, use the `maxvalue` ([1057](#)) function instead.

Errors: None.

See also: `maxvalue` ([1057](#)), `minvalue` ([1061](#)), `minIntValue` ([1060](#))

Listing: `./examples/mathex/ex25.pp`

Program Example25;

{ Program to demonstrate the MaxIntValue function. }

```
{ Make sore integer is 32 bit}
{$mode objfpc}
```

Uses math;

Type

TExArray = **Array**[1..100] **of** Integer;

Var

l : Integer;
ExArray : TExArray;

begin

Randomize;

for l:=**low**(exarray) **to high**(exarray) **do**

 ExArray[l]:=**Random**(l)-**Random**(100);

WriteLn(MaxIntValue(ExArray));

end.

63.13.63 MaxValue

Synopsis: Return largest value in array.

Declaration:

```
function MaxValue(const data: Array of Single) : Single
function MaxValue(const data: PSingle; const N: Integer) : Single
function MaxValue(const data: Array of Double) : Double
function MaxValue(const data: PDouble; const N: Integer) : Double
function MaxValue(const data: Array of Extended) : Extended
function MaxValue(const data: PExtended; const N: Integer) : Extended
function MaxValue(const data: Array of Integer) : Integer
function MaxValue(const data: PInteger; const N: Integer) : Integer
```

Visibility: default

Description: Maxvalue returns the largest value in the data array with integer or float values. The return value has the same type as the elements of the array.

The third and fourth forms accept a pointer to an array of N integer or float values.

Errors: None.

See also: maxIntValue ([1056](#)), minvalue ([1061](#)), minIntValue ([1060](#))

Listing: ./examples/mathex/ex26.pp

program Example26;

{ Program to demonstrate the MaxValue function. }

```
{ Make sore integer is 32 bit}
{$mode objfpc}
```

uses math;

var i:1..100;

 f_array:array[1..100] **of** Float;

 i_array:array[1..100] **of** Integer;

 Pf_array:Pfloat;

```

    Pi_array:Pinteger;

begin
    randomize;

    Pf_array:=@f_array[1];
    Pi_array:=@i_array[1];

    for i:=low(f_array) to high(f_array) do
        f_array[i]:=(random-random)*100;
    for i:=low(i_array) to high(i_array) do
        i_array[i]:=random(l)-random(100);

    Writeln('Max Float : ',MaxValue(f_array):8:4);
    Writeln('Max Float (b) : ',MaxValue(Pf_array,100):8:4);
    Writeln('Max Integer : ',MaxValue(i_array):8);
    Writeln('Max Integer (b) : ',MaxValue(Pi_array,100):8);
end.

```

63.13.64 Mean

Synopsis: Return mean value of array.

Declaration:

```

function Mean(const data: Array of Single) : Float
function Mean(const data: PSingle; const N: LongInt) : Float
function Mean(const data: Array of Double) : Float
function Mean(const data: PDouble; const N: LongInt) : Float
function Mean(const data: Array of Extended) : Float
function Mean(const data: PExtended; const N: LongInt) : Float
function Mean(const data: PInt64; const N: LongInt) : Float
function Mean(const data: Array of Int64) : Float
function Mean(const data: PInteger; const N: LongInt) : Float
function Mean(const data: Array of Integer) : Float

```

Visibility: default

Description: Mean returns the average value of data. The second form accepts a pointer to an array of N values.

Errors: None.

See also: meanandstddev ([1059](#)), momentskewkurtosis ([1062](#)), sum ([1076](#))

Listing: ./examples/mathex/ex27.pp

Program Example27;

```

{ Program to demonstrate the Mean function. }
{ @ should return typed pointer }
{$T+}
Uses math;

```

Type

```

TExArray = Array[1..100] of Float;

```

Var

```

l : Integer;
ExArray : TExArray;

```

```

begin
  Randomize;
  for I:=low(ExArray) to high(ExArray) do
    ExArray[i]:=(Random-Random)*100;
  Writeln('Max : ',MaxValue(ExArray):8:4);
  Writeln('Min : ',MinValue(ExArray):8:4);
  Writeln('Mean : ',Mean(ExArray):8:4);
  Writeln('Mean (b) : ',Mean(PExtended(@ExArray[1]),100):8:4);
end.

```

63.13.65 MeanAndStdDev

Synopsis: Return mean and standard deviation of array.

Declaration:

```

procedure MeanAndStdDev(const data: Array of Single; var mean: Float;
                        var stddev: Float)
procedure MeanAndStdDev(const data: PSingle; const N: LongInt;
                        var mean: Float; var stddev: Float)
procedure MeanAndStdDev(const data: Array of Double; var mean: Float;
                        var stddev: Float)
procedure MeanAndStdDev(const data: PDouble; const N: LongInt;
                        var mean: Float; var stddev: Float)
procedure MeanAndStdDev(const data: Array of Extended; var mean: Float;
                        var stddev: Float)
procedure MeanAndStdDev(const data: PExtended; const N: LongInt;
                        var mean: Float; var stddev: Float)

```

Visibility: default

Description: `meanandstddev` calculates the mean and standard deviation of `data` and returns the result in `mean` and `stddev`, respectively. `Stddev` is zero if there is only one value. The second form accepts a pointer to an array of `N` values.

Errors: None.

See also: `mean` ([1058](#)), `sum` ([1076](#)), `sumofsquares` ([1077](#)), `momentskewkurtosis` ([1062](#))

Listing: `./examples/mathex/ex28.pp`

Program Example28;

```

{ Program to demonstrate the Meanandstddev function. }
{ @ should return typed pointer }
{$T+}

```

Uses math;

Type

TExArray = **Array**[1..100] **of** Extended;

Var

```

I : Integer;
ExArray : TExArray;
Mean,stddev : Extended;

```

begin

```

Randomize;
for i:=low(ExArray) to high(ExArray) do
  ExArray[i]:=(Random-Random)*100;
MeanAndStdDev(ExArray,Mean,StdDev);
Writeln('Mean : ',Mean:8:4);
Writeln('StdDev : ',StdDev:8:4);
MeanAndStdDev(@ExArray[1],100,Mean,StdDev);
Writeln('Mean (b) : ',Mean:8:4);
Writeln('StdDev (b) : ',StdDev:8:4);
end.

```

63.13.66 Min

Synopsis: Return smallest of two values.

Declaration: `function Min(a: Integer; b: Integer) : Integer; Overload`
`function Min(a: Int64; b: Int64) : Int64; Overload`
`function Min(a: QWord; b: QWord) : QWord; Overload`
`function Min(a: Single; b: Single) : Single; Overload`
`function Min(a: Double; b: Double) : Double; Overload`
`function Min(a: Extended; b: Extended) : Extended; Overload`

Visibility: default

Description: `Min` returns the smallest of the two values in the `a` and `b` arguments. The overloaded variants use the same type for both of the compared arguments. The type for the first argument determines the overloaded function used for the comparison.

Errors: None.

See also: `max` ([1056](#)), `minIntValue` ([1060](#)), `minValue` ([1061](#))

Listing: `./examples/mathex/ex29.pp`

Program Example29;

{ Program to demonstrate the min function. }

Uses math;

Var

A,B : Cardinal;

begin

A:=1;b:=2;

writeln(min(a,b));

end.

63.13.67 MinIntValue

Synopsis: Return smallest value in integer array.

Declaration: `function MinIntValue(const Data: Array of Integer) : Integer`

Visibility: default

Description: `MinIntValue` returns the smallest value in the `Data` array.

This function is provided for Delphi compatibility, use `minvalue` instead.

Errors: None

See also: `minvalue` (1061), `maxIntValue` (1056), `maxvalue` (1057)

Listing: `./examples/mathex/ex30.pp`

Program Example30;

{ Program to demonstrate the MinIntValue function. }

{ Make sure integer is 32 bit }
{ \$mode objfpc }

Uses math;

Type

`TExArray = Array[1..100] of Integer;`

Var

`I : Integer;`
`ExArray : TExArray;`

begin

`Randomize;`

`for I:=low(ExArray) to high(ExArray) do`
`ExArray[I]:=Random(I)-Random(100);`
`WriteLn(MinIntValue(ExArray));`

end.

63.13.68 MinValue

Synopsis: Return smallest value in array.

Declaration:

```
function MinValue(const data: Array of Single) : Single
function MinValue(const data: PSingle; const N: Integer) : Single
function MinValue(const data: Array of Double) : Double
function MinValue(const data: PDouble; const N: Integer) : Double
function MinValue(const data: Array of Extended) : Extended
function MinValue(const data: PExtended; const N: Integer) : Extended
function MinValue(const data: Array of Integer) : Integer
function MinValue(const Data: PInteger; const N: Integer) : Integer
```

Visibility: default

Description: `Minvalue` returns the smallest value in the `data` array with integer or float values. The return value has the same type as the elements of the array.

The third and fourth forms accept a pointer to an array of `N` integer or float values.

Errors: None.

See also: `maxIntValue` (1056), `maxvalue` (1057), `minIntValue` (1060)

Listing: `./examples/mathex/ex31.pp`

```

program Example31;

{ Program to demonstrate the MinValue function. }

{ Make sure integer is 32 bit}
{$mode objfpc}

uses math;

var i:1..100;
    f_array:array[1..100] of Float;
    i_array:array[1..100] of Integer;
    Pf_array:Pfloat;
    Pi_array:Pinteger;

begin
    randomize;

    Pf_array:=@f_array[1];
    Pi_array:=@i_array[1];

    for i:=low(f_array) to high(f_array) do
        f_array[i]:=(random-random)*100;
    for i:=low(i_array) to high(i_array) do
        i_array[i]:=random(1)-random(100);

    Writeln('Min Float : ',MinValue(f_array):8:4);
    Writeln('Min Float (b) : ',MinValue(Pf_array,100):8:4);
    Writeln('Min Integer : ',MinValue(i_array):8);
    Writeln('Min Integer (b) : ',MinValue(Pi_array,100):8);
end.

```

63.13.69 modulus(Float,Float):Float

Synopsis: Floating point modulo.

Declaration: `operator mod(const a: Float; const b: Float) : Float`

Visibility: default

Description: Modulus is the floating-point equivalent of the modulo operation `a mod b`. It returns the result of

`a-b * Int (a/b)`

Errors: `b` may not be zero, but no check is performed.

See also: `FMod` ([1047](#))

63.13.70 MomentSkewKurtosis

Synopsis: Return 4 first moments of distribution.

Declaration: `procedure MomentSkewKurtosis(const data: Array of Single; out m1: Float; out m2: Float; out m3: Float; out m4: Float; out skew: Float; out kurtosis: Float)`

```

procedure MomentSkewKurtosis(const data: PSingle; const N: Integer;
                             out m1: Float; out m2: Float;
                             out m3: Float; out m4: Float;
                             out skew: Float; out kurtosis: Float)
procedure MomentSkewKurtosis(const data: Array of Double;
                             out m1: Float; out m2: Float;
                             out m3: Float; out m4: Float;
                             out skew: Float; out kurtosis: Float)
procedure MomentSkewKurtosis(const data: PDouble; const N: Integer;
                             out m1: Float; out m2: Float;
                             out m3: Float; out m4: Float;
                             out skew: Float; out kurtosis: Float)
procedure MomentSkewKurtosis(const data: Array of Extended;
                             out m1: Float; out m2: Float;
                             out m3: Float; out m4: Float;
                             out skew: Float; out kurtosis: Float)
procedure MomentSkewKurtosis(const data: PExtended; const N: Integer;
                             out m1: Float; out m2: Float;
                             out m3: Float; out m4: Float;
                             out skew: Float; out kurtosis: Float)

```

Visibility: default

Description: `momentskewkurtosis` calculates the 4 first moments of the distribution of value in data and returns them in `m1,m2,m3` and `m4`, as well as the skew and kurtosis.

Errors: None.

See also: [mean \(1058\)](#), [meanandstddev \(1059\)](#)

Listing: `./examples/mathex/ex32.pp`

```

program Example32;

{ Program to demonstrate the momentskewkurtosis function. }

uses math;

var distarray:array[1..1000] of float;
    l:longint;
    m1,m2,m3,m4,skew,kurtosis:float;

begin
  randomize;
  for l:=low(distarray) to high(distarray) do
    distarray[l]:=random;
  momentskewkurtosis(DistArray,m1,m2,m3,m4,skew,kurtosis);

  Writeln ('1st moment : ',m1:8:6);
  Writeln ('2nd moment : ',m2:8:6);
  Writeln ('3rd moment : ',m3:8:6);
  Writeln ('4th moment : ',m4:8:6);
  Writeln ('Skew : ',skew:8:6);
  Writeln ('kurtosis : ',kurtosis:8:6);
end.

```

63.13.71 Norm

Synopsis: Return Euclidean norm.

Declaration: `function Norm(const data: Array of Single) : Float`
`function Norm(const data: PSingle; const N: Integer) : Float`
`function Norm(const data: Array of Double) : Float`
`function Norm(const data: PDouble; const N: Integer) : Float`
`function Norm(const data: Array of Extended) : Float`
`function Norm(const data: PExtended; const N: Integer) : Float`

Visibility: default

Description: `Norm` calculates the Euclidean norm of the array of data. This equals `sqrt (sumofsquares (data) )`.
 The second form accepts a pointer to an array of N values.

Errors: None.

See also: `sumofsquares` ([1077](#))

Listing: `./examples/mathex/ex33.pp`

```
program Example33;

{ Program to demonstrate the norm function. }

uses math;

var v:array[1..10] of Float;
    l:1..10;

begin
  for l:=low(v) to high(v) do
    v[l]:=random;
  writeln(norm(v));
end.
```

63.13.72 NumberOfPeriods

Synopsis: Calculate the number of periods for an investment.

Declaration: `function NumberOfPeriods(ARate: Float; APayment: Float;`
`APresentValue: Float; AFutureValue: Float;`
`APaymentTime: TPaymentTime) : Float`

Visibility: default

Description: `NumberOfPeriods` calculates the number of periods (n) needed to obtain future value of an investment in the cash flow formula (see `CashFlowFunctions` ([1029](#))). The function result is the number of periods a payment `APayment` (PMT) must be paid in order to obtain a future value `AFutureValue` for an investment of a start value `APresentValue` (PV), where `APayment` (PMT) is invested at a rate `ARate` (q).

The `APaymentTime` parameter determines whether the investment (payment) is an ordinary annuity or an annuity due: `ptEndOfPeriod` `NumberOfPeriods` must be used if payments are at the end of each period. If the payments are at the beginning of the periode, `ptStartOfPeriod` must be used.

See also: `FutureValue` ([1048](#)), `InterestRate` ([1051](#)), `Payment` ([1065](#)), `PresentValue` ([1068](#)), `TPaymentTime` ([1034](#)), `CashFlowFunctions` ([1029](#))

63.13.73 Payment

Synopsis: Calculate the payment for an investment.

Declaration: `function Payment (ARate: Float; NPeriods: Integer; APresentValue: Float;
AFutureValue: Float; APaymentTime: TPaymentTime) : Float`

Visibility: default

Description: `Payment` calculates the amount that must be payed (PMT) during number of periods (n) needed to obtain future value of an investment in the cash flow formula (see `CashFlowFunctions` (1029)). The function result is the amount (PMT) that must be paid in order to obtain a future value `AFutureValue` for an investment of a start value `APresentValue` (PV), where the amount must be payed `NPeriods` (PMT) and the interest rate is `ARate` (q).

The `APaymentTime` parameter determines whether the investment (payment) is an ordinary annuity or an annuity due: `ptEndOfPeriod` `NumberOfPeriods` must be used if payments are at the end of each period. If the payments are at the beginning of the periode, `ptStartOfPeriod` must be used.

See also: `FutureValue` (1048), `InterestRate` (1051), `NumberOfPeriods` (1064), `PresentValue` (1068), `TPaymentTime` (1034), `CashFlowFunctions` (1029)

63.13.74 PopnStdDev

Synopsis: Return Population standard deviation.

Declaration: `function PopnStdDev(const data: Array of Single) : Float
function PopnStdDev(const data: PSingle; const N: Integer) : Float
function PopnStdDev(const data: Array of Double) : Float
function PopnStdDev(const data: PDouble; const N: Integer) : Float
function PopnStdDev(const data: Array of Extended) : Float
function PopnStdDev(const data: PExtended; const N: Integer) : Float`

Visibility: default

Description: `Popnstddev` returns the square root of the population variance of the values in the `Data` array. It returns zero if there is only one value.

The second form of this function accepts a pointer to an array of N values.

Errors: None.

See also: `popnvariance` (1066), `mean` (1058), `meanandstddev` (1059), `stddev` (1075), `momentskewkurtosis` (1062)

Listing: `./examples/mathex/ex35.pp`

Program `Example35;`

```
{ Program to demonstrate the PopnStdDev function. }  
{ @ should return typed pointer }  
{ $T+ }
```

Uses `Math;`

Type

`TExArray` = **Array**[1..100] of `Float`;

```

Var
  I : Integer;
  ExArray : TExArray;

begin
  Randomize;
  for I:=low(ExArray) to high(ExArray) do
    ExArray[i]:=(Random-Random)*100;
  Writeln('Max : ',MaxValue(ExArray):8:4);
  Writeln('Min : ',MinValue(ExArray):8:4);
  Writeln('Pop. stddev. : ',PopnStdDev(ExArray):8:4);
  Writeln('Pop. stddev. (b) : ',PopnStdDev(@ExArray[1],100):8:4);
end.

```

63.13.75 PopnVariance

Synopsis: Return population variance.

Declaration: `function PopnVariance(const data: PSingle; const N: Integer) : Float`
`function PopnVariance(const data: Array of Single) : Float`
`function PopnVariance(const data: PDouble; const N: Integer) : Float`
`function PopnVariance(const data: Array of Double) : Float`
`function PopnVariance(const data: PExtended; const N: Integer) : Float`
`function PopnVariance(const data: Array of Extended) : Float`

Visibility: default

Description: `Popnvariance` the population variance of the values in the `Data` array. It returns zero if there is only one value.

The second form of this function accepts a pointer to an array of `N` values.

Errors: None.

See also: `popnstddev` ([1065](#)), `mean` ([1058](#)), `meanandstddev` ([1059](#)), `stddev` ([1075](#)), `momentskewkurtosis` ([1062](#))

Listing: `./examples/mathex/ex36.pp`

Program Example36;

```

{ Program to demonstrate the PopnVariance function. }
{ @ should return typed pointer }
{$T+}

```

Uses math;

```

Var
  I : Integer;
  ExArray : Array[1..100] of Float;

```

```

begin
  Randomize;
  for I:=low(ExArray) to high(ExArray) do
    ExArray[i]:=(Random-Random)*100;
  Writeln('Max : ',MaxValue(ExArray):8:4);
  Writeln('Min : ',MinValue(ExArray):8:4);
  Writeln('Pop. var. : ',PopnVariance(ExArray):8:4);
  Writeln('Pop. var. (b) : ',PopnVariance(@ExArray[1],100):8:4);
end.

```

63.13.76 Power

Synopsis: Return real power.

Declaration: `function Power(base: Float; exponent: Float) : Float`

Visibility: default

Description: `power` raises `base` to the power of `exponent`. This is equivalent to `exp(ln(base)*exponent)`. However, for integer `exponent` values `intpower` (1051) is called. Otherwise `base` must be non-negative.

Errors: Any number of floating point math exceptions may be raised, if they are not masked by `SetExceptionMask` (1072).

See also: `intpower` (1051)

Listing: `./examples/mathex/ex34.pp`

Program Example34;

{ Program to demonstrate the power function. }

Uses Math;

procedure dopower(x,y : float);

begin

`writeln(x:8:6,'^',y:8:6,' = ',power(x,y):8:6)`

end;

begin

`dopower(2,2);`

`dopower(2,-2);`

`dopower(2,0.0);`

end.

63.13.77 power(Float,Float):Float

Synopsis: Raise base to the power exponent.

Declaration: `operator *(base: Float; exponent: Float) : Float`

Visibility: default

Description: `Power` raises `base` to the power `exponent`, i.e., it calculates $\text{base}^{\text{exponent}}$.

63.13.78 power(Int64,Int64):Int64

Synopsis: Raise base to the power exponent.

Declaration: `operator *(base: Int64; exponent: Int64) : Int64`

Visibility: default

Description: `Power` raises `base` to the power `exponent`, i.e., it calculates $\text{base}^{\text{exponent}}$.

63.13.79 PresentValue

Synopsis: Calculate the present value given the future value of an investment.

Declaration: `function PresentValue(ARate: Float; NPeriods: Integer; APayment: Float;
AFutureValue: Float; APaymentTime: TPaymentTime)
: Float`

Visibility: default

Description: `PresentValue` calculates the present (start) value of an investment (PV) in the cash flow formula (see `CashFlowFunctions` (1029)). The function result is the start value of an investment, when the future value is `AFutureValue` (FV) and `APayment` (PMT) is invested for `NPeriods` (n) at the rate of `ARate` (q) per period.

The `APaymentTime` parameter determines whether the investment (payment) is an ordinary annuity or an annuity due: `ptEndOfPeriod` must be used if payments are at the end of each period. If the payments are at the beginning of the periode, `ptStartOfPeriod` must be used.

See also: `FutureValue` (1048), `InterestRate` (1051), `NumberOfPeriods` (1064), `Payment` (1065), `TPaymentTime` (1034), `CashFlowFunctions` (1029)

63.13.80 RadToCycle

Synopsis: Convert radians angle to cycle angle.

Declaration: `function RadToCycle(const Rads: Single) : Single
function RadToCycle(const Rads: Double) : Double
function RadToCycle(const Rads: Extended) : Extended`

Visibility: default

Description: `RadtoCycle` converts its argument `rad` (an angle expressed in radians) to an angle in cycles.
(1 cycle equals 2 pi radians)

Errors: None.

See also: `degtograd` (1044), `degtorad` (1045), `radtoDeg` (1069), `radtoGrad` (1069), `cycletorad` (1043)

Listing: `./examples/mathex/ex37.pp`

Program Example37;

{ Program to demonstrate the radtoCycle function. }

Uses math;

begin
 writeln(radtoCycle(2*pi):8:6);
 writeln(radtoCycle(pi):8:6);
 writeln(radtoCycle(pi/2):8:6);
end.

63.13.81 RadToDeg

Synopsis: Convert radians angle to degrees angle.

Declaration: `function RadToDeg(rad: Float) : Float`

Visibility: default

Description: `RadToDeg` converts its argument `rad` (an angle expressed in radians) to an angle in degrees. (180 degrees equals pi radians)

Errors: None.

See also: `degtograd` ([1044](#)), `degtorad` ([1045](#)), `radtoCycle` ([1068](#)), `radtoGrad` ([1069](#)), `cycletorad` ([1043](#))

Listing: `./examples/mathex/ex38.pp`

Program Example38;

{ Program to demonstrate the radtoDeg function. }

Uses math;

begin

`writeln(radtoDeg(2*pi):8:6);`

`writeln(radtoDeg(pi):8:6);`

`writeln(radtoDeg(pi/2):8:6);`

end.

63.13.82 RadToGrad

Synopsis: Convert radians angle to grads angle.

Declaration: `function RadToGrad(rad: Float) : Float`

Visibility: default

Description: `RadToGrad` converts its argument `rad` (an angle expressed in radians) to an angle in grads. (200 grads equals pi radians)

Errors: None.

See also: `degtograd` ([1044](#)), `degtorad` ([1045](#)), `radtoCycle` ([1068](#)), `radtoDeg` ([1069](#)), `cycletorad` ([1043](#))

Listing: `./examples/mathex/ex39.pp`

Program Example39;

{ Program to demonstrate the radtoGrad function. }

Uses math;

begin

`writeln(radtoGrad(2*pi):8:6);`

`writeln(radtoGrad(pi):8:6);`

`writeln(radtoGrad(pi/2):8:6);`

end.

63.13.83 RandG

Synopsis: Return gaussian distributed random number.

Declaration: `function RandG(mean: Float; stddev: Float) : Float`

Visibility: default

Description: `randg` returns a random number which - when produced in large quantities - has a Gaussian distribution with mean `mean` and standarddeviation `stddev`.

Errors: None.

See also: `mean` ([1058](#)), `stddev` ([1075](#)), `meanandstddev` ([1059](#))

Listing: `./examples/mathex/ex40.pp`

Program Example40;

{ Program to demonstrate the randg function. }

Uses Math;

Var

`I : Integer;`
`ExArray : Array[1..10000] of Float;`
`Mean, stddev : Float;`

begin

Randomize;
for `I:=low(ExArray) to high(ExArray)` **do**
`ExArray[i]:=Randg(1,0.2);`
`MeanAndStdDev(ExArray,Mean,StdDev);`
Writeln('Mean : ',Mean:8:4);
Writeln('StdDev : ',StdDev:8:4);
end.

63.13.84 RandomFrom

Synopsis: Return a random element of an array of numbers.

Declaration: `function RandomFrom(const AValues: Array of Double) : Double; Overload`
`function RandomFrom(const AValues: Array of Integer) : Integer`
`; Overload`
`function RandomFrom(const AValues: Array of Int64) : Int64; Overload`

Visibility: default

Description: `RandomFrom` returns a random element from the array `AValues`. The return value has the same type as the type of the array elements.

See also: `#rtl.system.Random` ([1518](#)), `RandomRange` ([1070](#))

63.13.85 RandomRange

Synopsis: Return a random number in a range.

63.13.88 Sec

Synopsis: Alias for `secant`.

Declaration: `function Sec(x: Float) : Float`

Visibility: default

Description: `sec` is an alias for the `secant` ([1072](#)) function.

See also: `secant` ([1072](#))

63.13.89 Secant

Synopsis: Calculate `secant`.

Declaration: `function Secant(x: Float) : Float`

Visibility: default

Description: `Secant` calculates the `secant` ($1/\cos(x)$) of its argument `x`.

Errors: If 90 or 270 degrees is specified, an exception will be raised.

See also: `cosecant` ([1041](#))

63.13.90 SecH

Declaration: `function SecH(const X: Single) : Single`
`function SecH(const X: Double) : Double`
`function SecH(const X: Extended) : Extended`

Visibility: default

63.13.91 SetExceptionMask

Synopsis: Set the Floating Point Unit exception mask.

Declaration: `function SetExceptionMask(const Mask: TFPUEExceptionMask)`
`: TFPUEExceptionMask`

Visibility: default

Description: Set the Floating Point Unit exception mask.

63.13.92 SetPrecisionMode

Synopsis: Set the Floating Point Unit precision mode.

Declaration: `function SetPrecisionMode(const Precision: TFPUPrecisionMode)`
`: TFPUPrecisionMode`

Visibility: default

Description: Set the Floating Point Unit precision mode.

63.13.93 SetRoundMode

Synopsis: Set the Floating Point Unit rounding mode.

Declaration: `function SetRoundMode(const RoundMode: TFPURoundingMode)
: TFPURoundingMode`

Visibility: default

Description: `SetRoundMode` sets the floating point unit rounding mode. It also returns the previous rounding mode.

rmNearest Round to nearest integer.

rmDown Round to biggest integer smaller than value.

rmUp Round to smallest integer larger than value.

rmTruncate Cut off fractional part.

See also: `GetRoundMode` ([1049](#)), `TFPURoundingMode` ([1034](#))

63.13.94 Sign

Synopsis: Return sign of argument.

Declaration: `function Sign(const AValue: Integer) : TValueSign; Overload
function Sign(const AValue: Int64) : TValueSign; Overload
function Sign(const AValue: Single) : TValueSign; Overload
function Sign(const AValue: Double) : TValueSign; Overload
function Sign(const AValue: Extended) : TValueSign; Overload`

Visibility: default

Description: `Sign` returns the sign of it's argument, which can be an Integer, 64 bit integer, or a double. The returned value is an integer which is -1, 0 or 1, and can be used to do further calculations with.

63.13.95 SimpleRoundTo

Synopsis: Round to the specified number of digits (rounding up if needed).

Declaration: `function SimpleRoundTo(const AValue: Single;
const Digits: TRoundToRange) : Single
function SimpleRoundTo(const AValue: Double;
const Digits: TRoundToRange) : Double
function SimpleRoundTo(const AValue: Extended;
const Digits: TRoundToRange) : Extended`

Visibility: default

Description: `SimpleRoundTo` rounds the specified float `AValue` to the specified number of digits using symmetric arithmetic rounding (rounding up for positive or down for negative) and returns the result. This result is accurate to "10 to the power `Digits`". It uses the standard `Round` function for this.

Errors: An exception may occur if the value `AValue` is not inside a valid integer (or `Int64`) range.

See also: `TRoundToRange` ([1034](#)), `RoundTo` ([1071](#))

63.13.96 SinCos

Synopsis: Return sine and cosine of argument.

Declaration: `procedure SinCos(theta: single; out sinus: single; out cosinus: single)`
`procedure SinCos(theta: Double; out sinus: Double; out cosinus: Double)`
`procedure SinCos(theta: extended; out sinus: extended;`
`out cosinus: extended)`

Visibility: default

Description: `Sincos` calculates the sine and cosine of the angle `theta`, and returns the result in `sinus` and `cosinus`.

On Intel hardware, This calculation will be faster than making 2 calls to calculate the sine and cosine separately.

Errors: None.

See also: `arcsin` ([1037](#)), `arccos` ([1035](#))

Listing: `./examples/mathex/ex41.pp`

Program Example41;

{ Program to demonstrate the sincos function. }

Uses math;

Procedure dosincos(Angle : Float);

Var

Sine,Cosine : Float;

begin

sincos(angle,sine,cosine);

Write('Angle : ',Angle:8:6);

Write(' Sine : ',sine:8:6);

Write(' Cosine : ',cosine:8:6);

end;

begin

dosincos(**pi**);

dosincos(**pi**/2);

dosincos(**pi**/3);

dosincos(**pi**/4);

dosincos(**pi**/6);

end.

63.13.97 sinh

Synopsis: Return hyperbolic sine.

Declaration: `function sinh(x: Single) : Single`
`function sinh(x: Double) : Double`
`function sinh(x: Extended) : Extended`

Visibility: default

Description: `Sinh` returns the hyperbolic sine of its argument `x`.

See also: `cosh` (1041), `arsinh` (1039), `tanh` (1079), `artanh` (1039)

Listing: `./examples/mathex/ex42.pp`

Program Example42;

{ Program to demonstrate the sinh function. }

Uses math;

begin

writeln(sinh(0));

writeln(sinh(1));

writeln(sinh(-1));

end.

63.13.98 StdDev

Synopsis: Return standard deviation of data.

Declaration:

```
function StdDev(const data: Array of Single) : Float
function StdDev(const data: PSingle; const N: Integer) : Float
function StdDev(const data: Array of Double) : Float
function StdDev(const data: PDouble; const N: Integer) : Float
function StdDev(const data: Array of Extended) : Float
function StdDev(const data: PExtended; const N: Integer) : Float
```

Visibility: default

Description: `Stddev` returns the standard deviation of the values in `Data`. It returns zero if there is only one value.

The second form of the function accepts a pointer to an array of `N` values.

Errors: None.

See also: `mean` (1058), `meanandstddev` (1059), `variance` (1080), `totalvariance` (1080)

Listing: `./examples/mathex/ex43.pp`

Program Example40;

{ Program to demonstrate the stddev function. }

{ @ should return typed pointer }

{ \$T+ }

Uses Math;

Var

 I : Integer;

 ExArray : **Array**[1..10000] **of** Float;

begin

Randomize;

for I:=**low**(ExArray) **to** **high**(ExArray) **do**

 ExArray[I]:=Randg(1,0.2);

```

Writeln('StdDev : ',StdDev(ExArray):8:4);
Writeln('StdDev (b) : ',StdDev(@ExArray[1],10000):8:4);
end.

```

63.13.99 Sum

Synopsis: Return sum of values.

Declaration: `function Sum(const data: Array of Single) : Float`
`function Sum(const data: PSingle; const N: LongInt) : Float`
`function Sum(const data: Array of Double) : Float`
`function Sum(const data: PDouble; const N: LongInt) : Float`
`function Sum(const data: Array of Extended) : Float`
`function Sum(const data: PExtended; const N: LongInt) : Float`

Visibility: default

Description: `Sum` returns the sum of the values in the `data` array.

The second form of the function accepts a pointer to an array of `N` values.

Errors: None.

See also: `sumofsquares` ([1077](#)), `sumsandsquares` ([1077](#)), `totalvariance` ([1080](#)), `variance` ([1080](#))

Listing: `./examples/mathex/ex44.pp`

Program Example44;

```

{ Program to demonstrate the Sum function. }
{ @ should return typed pointer }
{$T+}

```

Uses math;

Var

```

I : 1..100;
ExArray : Array[1..100] of Float;

```

begin

```

  Randomize;
  for I:=low(ExArray) to high(ExArray) do
    ExArray[I]:=(Random-Random)*100;
  Writeln('Max : ',MaxValue(ExArray):8:4);
  Writeln('Min : ',MinValue(ExArray):8:4);
  Writeln('Sum : ',Sum(ExArray):8:4);
  Writeln('Sum (b) : ',Sum(@ExArray[1],100):8:4);

```

end.

63.13.100 SumInt

Synopsis: Return the sum of an array of integers.

Declaration: `function SumInt(const data: PInt64; const N: LongInt) : Int64`
`function SumInt(const data: Array of Int64) : Int64`
`function SumInt(const data: PInteger; const N: LongInt) : Int64`
`function SumInt(const data: Array of Integer) : Int64`

Visibility: default

Description: `SumInt` returns the sum of the `N` integers in the `Data` array, where this can be an open array or a pointer to an array.

Errors: An overflow may occur.

63.13.101 SumOfSquares

Synopsis: Return sum of squares of values.

Declaration: `function SumOfSquares(const data: Array of Single) : Float`
`function SumOfSquares(const data: PSingle; const N: Integer) : Float`
`function SumOfSquares(const data: Array of Double) : Float`
`function SumOfSquares(const data: PDouble; const N: Integer) : Float`
`function SumOfSquares(const data: Array of Extended) : Float`
`function SumOfSquares(const data: PExtended; const N: Integer) : Float`

Visibility: default

Description: `Sumofsquares` returns the sum of the squares of the values in the `data` array.

The second form of the function accepts a pointer to an array of `N` values.

Errors: None.

See also: `sum` (1076), `sumsandsquares` (1077), `totalvariance` (1080), `variance` (1080)

Listing: `./examples/mathex/ex45.pp`

Program Example45;

```
{ Program to demonstrate the SumOfSquares function. }
{ @ should return typed pointer }
{$T+}
```

Uses math;

Var

```
l : 1..100;
ExArray : Array[1..100] of Float;
```

begin

```
Randomize;
for l:=low(ExArray) to high(ExArray) do
  ExArray[l]:=(Random-Random)*100;
Writeln('Max : ',MaxValue(ExArray):8:4);
Writeln('Min : ',MinValue(ExArray):8:4);
Writeln('Sum squares : ',SumOfSquares(ExArray):8:4);
Writeln('Sum squares (b) : ',SumOfSquares(@ExArray[1],100):8:4);
end.
```

63.13.102 SumsAndSquares

Synopsis: Return sum and sum of squares of values.

Declaration:

```

procedure SumsAndSquares(const data: Array of Single; var sum: Float;
                        var sumofsquares: Float)
procedure SumsAndSquares(const data: PSingle; const N: Integer;
                        var sum: Float; var sumofsquares: Float)
procedure SumsAndSquares(const data: Array of Double; var sum: Float;
                        var sumofsquares: Float)
procedure SumsAndSquares(const data: PDouble; const N: Integer;
                        var sum: Float; var sumofsquares: Float)
procedure SumsAndSquares(const data: Array of Extended; var sum: Float;
                        var sumofsquares: Float)
procedure SumsAndSquares(const data: PExtended; const N: Integer;
                        var sum: Float; var sumofsquares: Float)

```

Visibility: default

Description: `sumsandsquares` calculates the sum of the values and the sum of the squares of the values in the `data` array and returns the results in `sum` and `sumofsquares`.

The second form of the function accepts a pointer to an array of `N` values.

Errors: None.

See also: `sum` ([1076](#)), `sumofsquares` ([1077](#)), `totalvariance` ([1080](#)), `variance` ([1080](#))

Listing: `./examples/mathex/ex46.pp`

Program Example45;

```

{ Program to demonstrate the SumOfSquares function. }
{ @ should return typed pointer }
{$T+}

```

Uses math;

Var

```

I : 1..100;
ExArray : Array[1..100] of Float;
s,ss : float;

```

begin

```

  Randomize;
  for I:=low(ExArray) to high(ExArray) do
    ExArray[i]:=(Random-Random)*100;
  Writeln('Max : ',MaxValue(ExArray):8:4);
  Writeln('Min : ',MinValue(ExArray):8:4);
  SumsAndSquares(ExArray,S,SS);
  Writeln('Sum : ',S:8:4);
  Writeln('Sum squares : ',SS:8:4);
  SumsAndSquares(@ExArray[1],100,S,SS);
  Writeln('Sum (b) : ',S:8:4);
  Writeln('Sum squares (b) : ',SS:8:4);

```

end.

63.13.103 Tan

Synopsis: Return tangent.

Declaration: `function Tan(x: Float) : Float`

Visibility: default

Description: `Tan` returns the tangent of x . The argument x must be in radians.

Errors: If x (normalized) is $\pi/2$ or $3\pi/2$ then an overflow will occur.

See also: `tanh` (1079), `arcsin` (1037), `sincos` (1074), `arccos` (1035)

Listing: ./examples/mathex/ex47.pp

Program Example47;

{ Program to demonstrate the Tan function. }

Uses math;

Procedure DoTan(Angle : Float);

begin

Write('Angle : ',RadToDeg(Angle):8:6);

Writeln(' Tangent : ',Tan(Angle):8:6);

end;

begin

 DoTan(0);

 DoTan(**Pi**);

 DoTan(**Pi**/3);

 DoTAn(**Pi**/4);

 DoTan(**Pi**/6);

end.

63.13.104 tanh

Synopsis: Return hyperbolic tangent.

Declaration: `function tanh(x: Single) : Single`
`function tanh(x: Double) : Double`
`function tanh(x: Extended) : Extended`

Visibility: default

Description: `Tanh` returns the hyperbolic tangent of x .

Errors: None.

See also: `arcsin` (1037), `sincos` (1074), `arccos` (1035)

Listing: ./examples/mathex/ex48.pp

Program Example48;

{ Program to demonstrate the Tanh function. }

Uses math;

begin

writeln(tanh(0));

writeln(tanh(1));

writeln(tanh(-1));

end.

63.13.105 TotalVariance

Synopsis: Return total variance of values.

Declaration: `function TotalVariance(const data: Array of Single) : Float`
`function TotalVariance(const data: PSingle; const N: Integer) : Float`
`function TotalVariance(const data: Array of Double) : Float`
`function TotalVariance(const data: PDouble; const N: Integer) : Float`
`function TotalVariance(const data: Array of Extended) : Float`
`function TotalVariance(const data: PExtended; const N: Integer) : Float`

Visibility: default

Description: `TotalVariance` returns the total variance of the values in the `data` array. It returns zero if there is only one value.

The second form of the function accepts a pointer to an array of `N` values.

Errors: None.

See also: [variance \(1080\)](#), [stddev \(1075\)](#), [mean \(1058\)](#)

Listing: `./examples/mathex/ex49.pp`

Program Example49;

```
{ Program to demonstrate the TotalVariance function. }
{ @ should return typed pointer }
{$T+}
```

Uses math;

Type

`TExArray = Array[1..100] of Float;`

Var

`I : Integer;`
`ExArray : TExArray;`
`TV : float;`

begin

```
Randomize;
for I:=1 to 100 do
  ExArray[I]:= (Random-Random)*100;
TV:=TotalVariance(ExArray);
Writeln('Total variance : ',TV:8:4);
TV:=TotalVariance(@ExArray[1],100);
Writeln('Total Variance (b) : ',TV:8:4);
```

end.

63.13.106 Variance

Synopsis: Return variance of values.

Declaration: `function Variance(const data: Array of Single) : Float`
`function Variance(const data: PSingle; const N: Integer) : Float`
`function Variance(const data: Array of Double) : Float`
`function Variance(const data: PDouble; const N: Integer) : Float`

```
function Variance(const data: Array of Extended) : Float
function Variance(const data: PExtended; const N: Integer) : Float
```

Visibility: default

Description: `Variance` returns the variance of the values in the `data` array. It returns zero if there is only one value.

The second form of the function accepts a pointer to an array of `N` values.

Errors: None.

See also: `totalvariance` ([1080](#)), `stddev` ([1075](#)), `mean` ([1058](#))

Listing: ./examples/mathex/ex50.pp

Program Example50;

```
{ Program to demonstrate the Variance function. }
{ @ should return typed pointer }
{$T+}
```

Uses math;

Var

```
l : 1..100;
ExArray : Array[1..100] of Float;
V : float;
```

begin

```
  Randomize;
  for l:=low(ExArray) to high(ExArray) do
    ExArray[l]:=(Random-Random)*100;
  V:=Variance(ExArray);
  Writeln('Variance : ',V:8:4);
  V:=Variance(@ExArray[1],100);
  Writeln('Variance (b) : ',V:8:4);
```

end.

Listing: ./examples/mathex/ex51.pp

Program Example51;

```
{
  Program to demonstrate the Variance function.
  It demonstrates the absence of large errors in the calculation.
}
```

Uses math;

const

```
Size = 1000000;
```

var

```
dataS: array of Single;
dataD: array of Double;
dataE: array of Extended;
i,n: longint;
```

begin

```
WriteLn('Each run should return a value near unity.');
```

```
WriteLn('Single:');
```

```
SetLength( dataS, Size );
```

```
for n := 1 to 4 do
```

```
begin
```

```
  for i := 0 to Size - 1 do
```

```
  begin
```

```
    dataS[i] := 10000000 + RandG(0,1);
```

```
  end;
```

```
  WriteLn( Math.Variance( dataS ):5:3 );
```

```
end;
```



```
WriteLn('Double:');
```

```
SetLength( dataD, Size );
```

```
for n := 1 to 4 do
```

```
begin
```

```
  for i := 0 to Size - 1 do
```

```
  begin
```

```
    dataD[i] := 10000000000000000 + RandG(0,1);
```

```
  end;
```

```
  WriteLn( Math.Variance( dataD ):5:3 );
```

```
end;
```



```
WriteLn('Extended:');
```

```
SetLength( dataE, Size );
```

```
for n := 1 to 4 do
```

```
begin
```

```
  for i := 0 to Size - 1 do
```

```
  begin
```

```
    dataE[i] := 10000000000000000000 + RandG(0,1);
```

```
  end;
```

```
  WriteLn( Math.Variance( dataE ):5:3 );
```

```
end;
```

```
end.
```

63.14 EInvalidArgument

63.14.1 Description

Exception raised when invalid arguments are passed to a function.

Chapter 64

Reference for unit 'matrix'

64.1 Used units

Table 64.1: Used units by unit 'matrix'

Name	Page
System	1335

64.2 Overview

The unit `matrix` is a unit that provides objects for the common two, three and four dimensional vectors matrices. These vectors and matrices are very common in computer graphics and are often implemented from scratch by programmers while every implementation provides exactly the same functionality.

It makes therefore sense to provide this functionality in the runtime library. This eliminates the need for programmers to reinvent the wheel and also allows libraries that use matrix operations to become more compatible.

The matrix unit does not provide n-dimensional matrices. The functionality needs of a general matrix unit varies from application to application; one can think of reduced memory usage tricks for matrices that only have data around the diagonal etc., desire for parallelization etc. etc. It is believed that programmers that do use n-dimensional matrices would not necessarily benefit from such a unit in the runtime library.

Design goals:

- Provide common dimensions, two three and four.
- Provide multiple floating point precisions, single, double, extended.
- Simple trivial binary representation; it is possible to typecast vectors into other implementations that use the same trivial representation.
- No dynamic memory management in the background. It must be possible to write expressions like `matrix A * B * C` without worrying about memory management.

Design decisions:

- Class object model is ruled out. The objects object model, without virtual methods, is suitable.
- Operator overloading is a good way to allow programmers to write matrix expressions.
- 3 dimensions * 3 precision means 9 vector and 9 matrix objects. Macro's have been used in the source to take care of this.

64.3 Constants, types and variables

64.3.1 Types

```
Tmatrix2_double_data = Array[0..1,0..1] of Double
```

This is the matrix internal data for a matrix. It uses a simple array structure so data from other libraries that define their own matrix type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tmatrix2_extended_data = Array[0..1,0..1] of extended
```

This is the matrix internal data for a matrix. It uses a simple array structure so data from other libraries that define their own matrix type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tmatrix2_single_data = Array[0..1,0..1] of single
```

This is the matrix internal data for a matrix. It uses a simple array structure so data from other libraries that define their own matrix type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tmatrix3_double_data = Array[0..2,0..2] of Double
```

This is the matrix internal data for a matrix. It uses a simple array structure so data from other libraries that define their own matrix type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tmatrix3_extended_data = Array[0..2,0..2] of extended
```

This is the matrix internal data for a matrix. It uses a simple array structure so data from other libraries that define their own matrix type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tmatrix3_single_data = Array[0..2,0..2] of single
```

This is the matrix internal data for a matrix. It uses a simple array structure so data from other libraries that define their own matrix type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tmatrix4_double_data = Array[0..3,0..3] of Double
```

This is the matrix internal data for a matrix. It uses a simple array structure so data from other libraries that define their own matrix type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tmatrix4_extended_data = Array[0..3,0..3] of extended
```

This is the matrix internal data for a matrix. It uses a simple array structure so data from other libraries that define their own matrix type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tmatrix4_single_data = Array[0..3,0..3] of single
```

This is the matrix internal data for a matrix. It uses a simple array structure so data from other libraries that define their own matrix type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tvector2_double_data = Array[0..1] of Double
```

This is the vector internal data for a vector. It uses a simple array structure so data from other libraries that define their own vector type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tvector2_extended_data = Array[0..1] of extended
```

This is the vector internal data for a vector. It uses a simple array structure so data from other libraries that define their own vector type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tvector2_single_data = Array[0..1] of single
```

This is the vector internal data for a vector. It uses a simple array structure so data from other libraries that define their own vector type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tvector3_double_data = Array[0..2] of Double
```

This is the vector internal data for a vector. It uses a simple array structure so data from other libraries that define their own vector type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tvector3_extended_data = Array[0..2] of extended
```

This is the vector internal data for a vector. It uses a simple array structure so data from other libraries that define their own vector type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tvector3_single_data = Array[0..2] of single
```

This is the vector internal data for a vector. It uses a simple array structure so data from other libraries that define their own vector type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tvector4_double_data = Array[0..3] of Double
```

This is the vector internal data for a vector. It uses a simple array structure so data from other libraries that define their own vector type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tvector4_extended_data = Array[0..3] of extended
```

This is the vector internal data for a vector. It uses a simple array structure so data from other libraries that define their own vector type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

```
Tvector4_single_data = Array[0..3] of single
```

This is the vector internal data for a vector. It uses a simple array structure so data from other libraries that define their own vector type as a simple array structure can simply be moved to and from this data, or typecasted into it. As this is the only field in the object, the object itself can be used just as fine for typecasting purposes etc.

64.4 Procedures and functions

64.4.1 add(Tmatrix2_double,Double):Tmatrix2_double

Synopsis: Add scalar to two-dimensional double precision matrix.

Declaration: `operator +(const m: Tmatrix2_double; const x: Double) : Tmatrix2_double`

Visibility: default

Description: This operator allows you to add a scalar value to a matrix. The scalar is added to all elements of the matrix, the result is returned as a new vector.

64.4.2 add(Tmatrix2_double,Tmatrix2_double):Tmatrix2_double

Synopsis: Add two two-dimensional double precision matrices together.

Declaration: `operator +(const m1: Tmatrix2_double; const m2: Tmatrix2_double)
: Tmatrix2_double`

Visibility: default

Description: This operator allows you to add two two-dimensional double precision matrices together. A new matrix is returned with all elements of the two matrices added together.

64.4.3 add(Tmatrix2_extended,extended):Tmatrix2_extended

Synopsis: Add scalar to two-dimensional extended precision matrix.

Declaration: `operator +(const m: Tmatrix2_extended; const x: extended)
: Tmatrix2_extended`

Visibility: default

Description: This operator allows you to add a scalar value to a matrix. The scalar is added to all elements of the matrix, the result is returned as a new vector.

64.4.4 add(Tmatrix2_extended,Tmatrix2_extended):Tmatrix2_extended

Synopsis: Add two two-dimensional extended precision matrices together.

Declaration: `operator +(const m1: Tmatrix2_extended; const m2: Tmatrix2_extended)
: Tmatrix2_extended`

Visibility: default

Description: This operator allows you to add two two-dimensional extended precision matrices together. A new matrix is returned with all elements of the two matrices added together.

64.4.5 add(Tmatrix2_single,single):Tmatrix2_single

Synopsis: Add scalar to two-dimensional single precision matrix.

Declaration: `operator +(const m: Tmatrix2_single; const x: single) : Tmatrix2_single`

Visibility: default

Description: This operator allows you to add a scalar value to a matrix. The scalar is added to all elements of the matrix, the result is returned as a new vector.

64.4.6 add(Tmatrix2_single,Tmatrix2_single):Tmatrix2_single

Synopsis: Add two two-dimensional single precision matrices together.

Declaration: `operator +(const m1: Tmatrix2_single; const m2: Tmatrix2_single)
: Tmatrix2_single`

Visibility: default

Description: This operator allows you to add two two-dimensional single precision matrices together. A new matrix is returned with all elements of the two matrices added together.

64.4.7 add(Tmatrix3_double,Double):Tmatrix3_double

Synopsis: Add scalar to three-dimensional double precision matrix.

Declaration: `operator +(const m: Tmatrix3_double; const x: Double) : Tmatrix3_double`

Visibility: default

Description: This operator allows you to add a scalar value to a matrix. The scalar is added to all elements of the matrix, the result is returned as a new vector.

64.4.8 add(Tmatrix3_double,Tmatrix3_double):Tmatrix3_double

Synopsis: Add two three-dimensional double precision matrices together.

Declaration: `operator +(const m1: Tmatrix3_double; const m2: Tmatrix3_double)
: Tmatrix3_double`

Visibility: default

Description: This operator allows you to add two three-dimensional double precision matrices together. A new matrix is returned with all elements of the two matrices added together.

64.4.9 add(Tmatrix3_extended,extended):Tmatrix3_extended

Synopsis: Add scalar to three-dimensional extended precision matrix.

Declaration: `operator +(const m: Tmatrix3_extended; const x: extended)
: Tmatrix3_extended`

Visibility: default

Description: This operator allows you to add a scalar value to a matrix. The scalar is added to all elements of the matrix, the result is returned as a new vector.

64.4.10 add(Tmatrix3_extended,Tmatrix3_extended):Tmatrix3_extended

Synopsis: Add two three-dimensional extended precision matrices together.

Declaration: `operator +(const m1: Tmatrix3_extended; const m2: Tmatrix3_extended)
: Tmatrix3_extended`

Visibility: default

Description: This operator allows you to add two three-dimensional extended precision matrices together. A new matrix is returned with all elements of the two matrices added together.

64.4.11 add(Tmatrix3_single,single):Tmatrix3_single

Synopsis: Add scalar to three-dimensional single precision matrix.

Declaration: `operator +(const m: Tmatrix3_single; const x: single) : Tmatrix3_single`

Visibility: default

Description: This operator allows you to add a scalar value to a matrix. The scalar is added to all elements of the matrix, the result is returned as a new vector.

64.4.12 add(Tmatrix3_single,Tmatrix3_single):Tmatrix3_single

Synopsis: Add two three-dimensional single precision matrices together.

Declaration: `operator +(const m1: Tmatrix3_single; const m2: Tmatrix3_single)
: Tmatrix3_single`

Visibility: default

Description: This operator allows you to add two three-dimensional single precision matrices together. A new matrix is returned with all elements of the two matrices added together.

64.4.13 add(Tmatrix4_double,Double):Tmatrix4_double

Synopsis: Add scalar to four-dimensional double precision matrix.

Declaration: `operator +(const m: Tmatrix4_double; const x: Double) : Tmatrix4_double`

Visibility: default

Description: This operator allows you to add a scalar value to a matrix. The scalar is added to all elements of the matrix, the result is returned as a new vector.

64.4.14 add(Tmatrix4_double,Tmatrix4_double):Tmatrix4_double

Synopsis: Add two four-dimensional double precision matrices together.

Declaration: `operator +(const m1: Tmatrix4_double; const m2: Tmatrix4_double)
: Tmatrix4_double`

Visibility: default

Description: This operator allows you to add two four-dimensional double precision matrices together. A new matrix is returned with all elements of the two matrices added together.

64.4.15 add(Tmatrix4_extended,extended):Tmatrix4_extended

Synopsis: Add scalar to four-dimensional extended precision matrix.

Declaration: `operator +(const m: Tmatrix4_extended; const x: extended)
: Tmatrix4_extended`

Visibility: default

Description: This operator allows you to add a scalar value to a matrix. The scalar is added to all elements of the matrix, the result is returned as a new vector.

64.4.16 add(Tmatrix4_extended,Tmatrix4_extended):Tmatrix4_extended

Synopsis: Add two four-dimensional extended precision matrices together.

Declaration: `operator +(const m1: Tmatrix4_extended; const m2: Tmatrix4_extended)
: Tmatrix4_extended`

Visibility: default

Description: This operator allows you to add two four-dimensional extended precision matrices together. A new matrix is returned with all elements of the two matrices added together.

64.4.17 add(Tmatrix4_single,single):Tmatrix4_single

Synopsis: Add scalar to four-dimensional single precision matrix.

Declaration: `operator +(const m: Tmatrix4_single; const x: single) : Tmatrix4_single`

Visibility: default

Description: This operator allows you to add a scalar value to a matrix. The scalar is added to all elements of the matrix, the result is returned as a new vector.

64.4.18 add(Tmatrix4_single,Tmatrix4_single):Tmatrix4_single

Synopsis: Add two four-dimensional single precision matrices together.

Declaration: `operator +(const m1: Tmatrix4_single; const m2: Tmatrix4_single)
: Tmatrix4_single`

Visibility: default

Description: This operator allows you to add two four-dimensional single precision matrices together. A new matrix is returned with all elements of the two matrices added together.

64.4.19 add(Tvector2_double,Double):Tvector2_double

Synopsis: Add scalar to two-dimensional double precision vector.

Declaration: `operator +(const x: Tvector2_double; y: Double) : Tvector2_double`

Visibility: default

Description: This operator allows you to add a scalar value to a vector. The scalar is added to all elements of the vector, the result is returned as a new vector.

64.4.20 add(Tvector2_double,Tvector2_double):Tvector2_double

Synopsis: Add two-dimensional double precision vectors together.

Declaration: `operator +(const x: Tvector2_double; const y: Tvector2_double)
: Tvector2_double`

Visibility: default

Description: This operator allows you to add two two-dimensional vectors with double precision together. The result is a new vector which consists of the sums of the individual elements of the two vectors.

64.4.21 add(Tvector2_extended,extended):Tvector2_extended

Synopsis: Add scalar to two-dimensional extended precision vector.

Declaration: `operator +(const x: Tvector2_extended; y: extended) : Tvector2_extended`

Visibility: default

Description: This operator allows you to add a scalar value to a vector. The scalar is added to all elements of the vector, the result is returned as a new vector.

64.4.22 add(Tvector2_extended,Tvector2_extended):Tvector2_extended

Synopsis: Add two-dimensional extended precision vectors together.

Declaration: `operator +(const x: Tvector2_extended; const y: Tvector2_extended)
: Tvector2_extended`

Visibility: default

Description: This operator allows you to add two two-dimensional vectors with extended precision together. The result is a new vector which consists of the sums of the individual elements of the two vectors.

64.4.23 add(Tvector2_single,single):Tvector2_single

Synopsis: Add scalar to two-dimensional single precision vector.

Declaration: `operator +(const x: Tvector2_single; y: single) : Tvector2_single`

Visibility: default

Description: This operator allows you to add a scalar value to a vector. The scalar is added to all elements of the vector, the result is returned as a new vector.

64.4.24 add(Tvector2_single,Tvector2_single):Tvector2_single

Synopsis: Add two-dimensional single precision vectors together.

Declaration: `operator +(const x: Tvector2_single; const y: Tvector2_single)
: Tvector2_single`

Visibility: default

Description: This operator allows you to add two two-dimensional vectors with single precision together. The result is a new vector which consists of the sums of the individual elements of the two vectors.

64.4.25 add(Tvector3_double,Double):Tvector3_double

Synopsis: Add scalar to three-dimensional double precision vector.

Declaration: `operator +(const x: Tvector3_double; y: Double) : Tvector3_double`

Visibility: default

Description: This operator allows you to add a scalar value to a vector. The scalar is added to all elements of the vector, the result is returned as a new vector.

64.4.26 add(Tvector3_double,Tvector3_double):Tvector3_double

Synopsis: Add three-dimensional double precision vectors together.

Declaration: `operator +(const x: Tvector3_double; const y: Tvector3_double)
: Tvector3_double`

Visibility: default

Description: This operator allows you to add two three-dimensional vectors with double precision together. The result is a new vector which consists of the sums of the individual elements of the two vectors.

64.4.27 add(Tvector3_extended,extended):Tvector3_extended

Synopsis: Add scalar to three-dimensional extended precision vector.

Declaration: `operator +(const x: Tvector3_extended; y: extended) : Tvector3_extended`

Visibility: default

Description: This operator allows you to add a scalar value to a vector. The scalar is added to all elements of the vector, the result is returned as a new vector.

64.4.28 add(Tvector3_extended,Tvector3_extended):Tvector3_extended

Synopsis: Add three-dimensional extended precision vectors together.

Declaration: `operator +(const x: Tvector3_extended; const y: Tvector3_extended)
: Tvector3_extended`

Visibility: default

Description: This operator allows you to add two three-dimensional vectors with extended precision together. The result is a new vector which consists of the sums of the individual elements of the two vectors.

64.4.29 add(Tvector3_single,single):Tvector3_single

Synopsis: Add scalar to three-dimensional single precision vector.

Declaration: `operator +(const x: Tvector3_single; y: single) : Tvector3_single`

Visibility: default

Description: This operator allows you to add a scalar value to a vector. The scalar is added to all elements of the vector, the result is returned as a new vector.

64.4.30 add(Tvector3_single,Tvector3_single):Tvector3_single

Synopsis: Add three-dimensional extended precision vectors together.

Declaration: `operator +(const x: Tvector3_single; const y: Tvector3_single)
: Tvector3_single`

Visibility: default

Description: This operator allows you to add two three-dimensional vectors with single precision together. The result is a new vector which consists of the sums of the individual elements of the two vectors.

64.4.31 add(Tvector4_double,Double):Tvector4_double

Synopsis: Add scalar to four-dimensional double precision vector.

Declaration: `operator +(const x: Tvector4_double; y: Double) : Tvector4_double`

Visibility: default

Description: This operator allows you to add a scalar value to a vector. The scalar is added to all elements of the vector, the result is returned as a new vector.

64.4.32 add(Tvector4_double,Tvector4_double):Tvector4_double

Synopsis: Add four-dimensional double precision vectors together.

Declaration: `operator +(const x: Tvector4_double; const y: Tvector4_double)
: Tvector4_double`

Visibility: default

Description: This operator allows you to add two four-dimensional vectors with single precision together. The result is a new vector which consists of the sums of the individual elements of the two vectors.

64.4.33 add(Tvector4_extended,extended):Tvector4_extended

Synopsis: Add scalar to four-dimensional extended precision vector.

Declaration: `operator +(const x: Tvector4_extended; y: extended) : Tvector4_extended`

Visibility: default

Description: This operator allows you to add a scalar value to a vector. The scalar is added to all elements of the vector, the result is returned as a new vector.

64.4.34 add(Tvector4_extended,Tvector4_extended):Tvector4_extended

Synopsis: Add four-dimensional extended precision vectors together.

Declaration: `operator +(const x: Tvector4_extended; const y: Tvector4_extended) : Tvector4_extended`

Visibility: default

Description: This operator allows you to add two two-dimensional vectors with extended precision together. The result is a new vector which consists of the sums of the individual elements of the two vectors.

64.4.35 add(Tvector4_single,single):Tvector4_single

Synopsis: Add scalar to four-dimensional single precision vector.

Declaration: `operator +(const x: Tvector4_single; y: single) : Tvector4_single`

Visibility: default

Description: This operator allows you to add a scalar value to a vector. The scalar is added to all elements of the vector, the result is returned as a new vector.

64.4.36 add(Tvector4_single,Tvector4_single):Tvector4_single

Synopsis: Add four-dimensional single precision vectors together.

Declaration: `operator +(const x: Tvector4_single; const y: Tvector4_single) : Tvector4_single`

Visibility: default

Description: This operator allows you to add two four-dimensional vectors with single precision together. The result is a new vector which consists of the sums of the individual elements of the two vectors.

64.4.37 assign(Tmatrix2_double):Tmatrix2_extended

Synopsis: Allow assignment of two-dimensional double precision matrix to two-dimensional extended precision matrix.

Declaration: `operator :=(const v: Tmatrix2_double) : Tmatrix2_extended`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with double precision values wherever a two-dimensional matrix with extended precision is expected.

64.4.38 assign(Tmatrix2_double):Tmatrix2_single

Synopsis: Allow assignment of two-dimensional double precision matrix to two-dimensional single precision matrix.

Declaration: `operator :=(const v: Tmatrix2_double) : Tmatrix2_single`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with double precision values wherever a two-dimensional matrix with single precision is expected. Some accuracy is lost because of the conversion.

64.4.39 assign(Tmatrix2_double):Tmatrix3_double

Synopsis: Allow assignment of two-dimensional double precision matrix to three-dimensional double precision matrix.

Declaration: `operator :=(const v: Tmatrix2_double) : Tmatrix3_double`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with double precision values wherever a three-dimensional matrix with double precision is expected. The extra fields are set to 0.

64.4.40 assign(Tmatrix2_double):Tmatrix3_extended

Synopsis: Allow assignment of two-dimensional double precision matrix to three-dimensional extended precision matrix.

Declaration: `operator :=(const v: Tmatrix2_double) : Tmatrix3_extended`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with double precision values wherever a three-dimensional matrix with extended precision is expected. The extra fields are set to 0.

64.4.41 assign(Tmatrix2_double):Tmatrix3_single

Synopsis: Allow assignment of two-dimensional single precision matrix to three-dimensional single precision matrix.

Declaration: `operator :=(const v: Tmatrix2_double) : Tmatrix3_single`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with single precision values wherever a three-dimensional matrix with single precision is expected. The extra fields are set to 0 and some accuracy is lost because of the conversion.

64.4.42 assign(Tmatrix2_double):Tmatrix4_double

Synopsis: Allow assignment of two-dimensional double precision matrix to four-dimensional double precision matrix.

Declaration: `operator :=(const v: Tmatrix2_double) : Tmatrix4_double`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with double precision values wherever a four-dimensional matrix with double precision is expected. The extra fields are set to 0.

64.4.43 **assign(Tmatrix2_double):Tmatrix4_extended**

Synopsis: Allow assignment of two-dimensional double precision matrix to four-dimensional extended precision matrix.

Declaration: `operator :=(const v: Tmatrix2_double) : Tmatrix4_extended`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with double precision values wherever a four-dimensional matrix with extended precision is expected. The extra fields are set to 0.

64.4.44 **assign(Tmatrix2_double):Tmatrix4_single**

Synopsis: Allow assignment of two-dimensional double precision matrix to four-dimensional single precision matrix.

Declaration: `operator :=(const v: Tmatrix2_double) : Tmatrix4_single`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with double precision values wherever a four-dimensional matrix with single precision is expected. The extra fields are set to 0 and some precision is lost because of the conversion.

64.4.45 **assign(Tmatrix2_extended):Tmatrix2_double**

Synopsis: Allow assignment of two-dimensional extended precision matrix to two-dimensional double precision matrix.

Declaration: `operator :=(const v: Tmatrix2_extended) : Tmatrix2_double`

Visibility: default

Description: This operator allows you to use a two-dimensional two with extended precision values wherever a two-dimensional matrix with double precision is expected. Some accuracy is lost because of the conversion.

64.4.46 **assign(Tmatrix2_extended):Tmatrix2_single**

Synopsis: Allow assignment of two-dimensional extended precision matrix to two-dimensional single precision matrix.

Declaration: `operator :=(const v: Tmatrix2_extended) : Tmatrix2_single`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with extended precision values wherever a two-dimensional matrix with single precision is expected. Some accuracy is lost because of the conversion.

64.4.47 assign(Tmatrix2_extended):Tmatrix3_double

Synopsis: Allow assignment of two-dimensional extended precision matrix to three-dimensional double precision matrix.

Declaration: `operator :=(const v: Tmatrix2_extended) : Tmatrix3_double`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with extended precision values wherever a three-dimensional matrix with double precision is expected. The extra fields are set to 0 and some accuracy is lost because of the conversion.

64.4.48 assign(Tmatrix2_extended):Tmatrix3_extended

Synopsis: Allow assignment of two-dimensional extended precision matrix to three-dimensional extended precision matrix.

Declaration: `operator :=(const v: Tmatrix2_extended) : Tmatrix3_extended`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with extended precision values wherever a three-dimensional matrix with extended precision is expected. The extra fields are set to 0.

64.4.49 assign(Tmatrix2_extended):Tmatrix3_single

Synopsis: Allow assignment of two-dimensional extended precision matrix to three-dimensional single precision matrix.

Declaration: `operator :=(const v: Tmatrix2_extended) : Tmatrix3_single`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with extended precision values wherever a three-dimensional matrix with single precision is expected. The extra fields are set to 0 and some accuracy is lost because of the conversion.

64.4.50 assign(Tmatrix2_extended):Tmatrix4_double

Synopsis: Allow assignment of two-dimensional extended precision matrix to four-dimensional double precision matrix.

Declaration: `operator :=(const v: Tmatrix2_extended) : Tmatrix4_double`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with extended precision values wherever a four-dimensional matrix with double precision is expected. The extra fields are set to 0 and some accuracy is lost because of the conversion.

64.4.51 assign(Tmatrix2_extended):Tmatrix4_extended

Synopsis: Allow assignment of two-dimensional extended precision matrix to four-dimensional extended precision matrix.

Declaration: `operator :=(const v: Tmatrix2_extended) : Tmatrix4_extended`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with single precision values wherever a four-dimensional matrix with single precision is expected. The extra fields are set to 0.

64.4.52 assign(Tmatrix2_extended):Tmatrix4_single

Synopsis: Allow assignment of two-dimensional extended precision matrix to four-dimensional single precision matrix.

Declaration: `operator :=(const v: Tmatrix2_extended) : Tmatrix4_single`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with extended precision values wherever a four-dimensional matrix with single precision is expected. The extra fields are set to 0 and some precision is lost because of the conversion.

64.4.53 assign(Tmatrix2_single):Tmatrix2_double

Synopsis: Allow assignment of two-dimensional single precision matrix to two-dimensional double precision matrix.

Declaration: `operator :=(const v: Tmatrix2_single) : Tmatrix2_double`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with single precision values wherever a two-dimensional matrix with double precision is expected.

64.4.54 assign(Tmatrix2_single):Tmatrix2_extended

Synopsis: Allow assignment of two-dimensional single precision matrix to two-dimensional extended precision matrix.

Declaration: `operator :=(const v: Tmatrix2_single) : Tmatrix2_extended`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with single precision values wherever a two-dimensional matrix with extended precision is expected.

64.4.55 assign(Tmatrix2_single):Tmatrix3_double

Synopsis: Allow assignment of two-dimensional single precision matrix to three-dimensional double precision matrix.

Declaration: `operator :=(const v: Tmatrix2_single) : Tmatrix3_double`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with single precision values wherever a three-dimensional matrix with double precision is expected. The extra fields are set to 0.

64.4.56 assign(Tmatrix2_single):Tmatrix3_extended

Synopsis: Allow assignment of two-dimensional single precision matrix to three-dimensional extended precision matrix.

Declaration: `operator :=(const v: Tmatrix2_single) : Tmatrix3_extended`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with single precision values wherever a three-dimensional matrix with extended precision is expected. The extra fields are set to 0.

64.4.57 assign(Tmatrix2_single):Tmatrix3_single

Synopsis: Allow assignment of two-dimensional single precision matrix to three-dimensional single precision matrix.

Declaration: `operator :=(const v: Tmatrix2_single) : Tmatrix3_single`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with single precision values wherever a three-dimensional matrix with single precision is expected. The extra fields are set to 0.

64.4.58 assign(Tmatrix2_single):Tmatrix4_double

Synopsis: Allow assignment of two-dimensional single precision matrix to four-dimensional double precision matrix.

Declaration: `operator :=(const v: Tmatrix2_single) : Tmatrix4_double`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with single precision values wherever a four-dimensional matrix with double precision is expected. The extra fields are set to 0.

64.4.59 assign(Tmatrix2_single):Tmatrix4_extended

Synopsis: Allow assignment of two-dimensional single precision matrix to four-dimensional extended precision matrix.

Declaration: `operator :=(const v: Tmatrix2_single) : Tmatrix4_extended`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with single precision values wherever a four-dimensional matrix with extended precision is expected. The extra fields are set to 0.

64.4.60 assign(Tmatrix2_single):Tmatrix4_single

Synopsis: Allow assignment of two-dimensional single precision matrix to four-dimensional single precision matrix.

Declaration: `operator :=(const v: Tmatrix2_single) : Tmatrix4_single`

Visibility: default

Description: This operator allows you to use a two-dimensional matrix with single precision values wherever a four-dimensional matrix with single precision is expected. The extra fields are set to 0.

64.4.61 assign(Tmatrix3_double):Tmatrix2_double

Synopsis: Allow assignment of three-dimensional double precision matrix to two-dimensional double precision matrix.

Declaration: `operator :=(const v: Tmatrix3_double) : Tmatrix2_double`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with double precision values wherever a two-dimensional matrix with double precision is expected. The surplus fields are thrown away.

64.4.62 assign(Tmatrix3_double):Tmatrix2_extended

Synopsis: Allow assignment of three-dimensional double precision matrix to two-dimensional extended precision matrix.

Declaration: `operator :=(const v: Tmatrix3_double) : Tmatrix2_extended`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with double precision values wherever a two-dimensional matrix with extended precision is expected. The surplus fields are thrown away.

64.4.63 assign(Tmatrix3_double):Tmatrix2_single

Synopsis: Allow assignment of three-dimensional double precision matrix to two-dimensional single precision matrix.

Declaration: `operator :=(const v: Tmatrix3_double) : Tmatrix2_single`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with double precision values wherever a two-dimensional matrix with single precision is expected. The surplus fields are thrown away and some accuracy is lost because of the conversion.

64.4.64 assign(Tmatrix3_double):Tmatrix3_extended

Synopsis: Allow assignment of three-dimensional double precision matrix to three-dimensional extended precision matrix.

Declaration: `operator :=(const v: Tmatrix3_double) : Tmatrix3_extended`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with double precision values wherever a three-dimensional matrix with extended precision is expected.

64.4.65 assign(Tmatrix3_double):Tmatrix3_single

Synopsis: Allow assignment of three-dimensional double precision matrix to three-dimensional single precision matrix.

Declaration: `operator :=(const v: Tmatrix3_double) : Tmatrix3_single`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with double precision values wherever a three-dimensional matrix with single precision is expected. Some precision is lost because of the conversion.

64.4.66 **assign(Tmatrix3_double):Tmatrix4_double**

Synopsis: Allow assignment of three-dimensional double precision matrix to four-dimensional double precision matrix.

Declaration: `operator :=(const v: Tmatrix3_double) : Tmatrix4_double`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with double precision values wherever a four-dimensional matrix with double precision is expected.

64.4.67 **assign(Tmatrix3_double):Tmatrix4_extended**

Synopsis: Allow assignment of three-dimensional double precision matrix to four-dimensional extended precision matrix.

Declaration: `operator :=(const v: Tmatrix3_double) : Tmatrix4_extended`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with double precision values wherever a four-dimensional matrix with extended precision is expected.

64.4.68 **assign(Tmatrix3_double):Tmatrix4_single**

Synopsis: Allow assignment of three-dimensional double precision matrix to four-dimensional single precision matrix.

Declaration: `operator :=(const v: Tmatrix3_double) : Tmatrix4_single`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with double precision values wherever a four-dimensional matrix with single precision is expected. Some precision is lost because of the conversion.

64.4.69 **assign(Tmatrix3_extended):Tmatrix2_double**

Synopsis: Allow assignment of three-dimensional extended precision matrix to two-dimensional double precision matrix.

Declaration: `operator :=(const v: Tmatrix3_extended) : Tmatrix2_double`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with extended precision values wherever a two-dimensional matrix with double precision is expected. The surplus fields are thrown away and some accuracy is lost because of the conversion.

64.4.70 assign(Tmatrix3_extended):Tmatrix2_extended

Synopsis: Allow assignment of three-dimensional extended precision matrix to two-dimensional extended precision matrix.

Declaration: `operator :=(const v: Tmatrix3_extended) : Tmatrix2_extended`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with extended precision values wherever a two-dimensional matrix with extended precision is expected. The surplus fields are thrown away.

64.4.71 assign(Tmatrix3_extended):Tmatrix2_single

Synopsis: Allow assignment of three-dimensional extended precision matrix to two-dimensional single precision matrix.

Declaration: `operator :=(const v: Tmatrix3_extended) : Tmatrix2_single`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with extended precision values wherever a two-dimensional matrix with single precision is expected. The surplus fields are thrown away and some precision is lost because of the conversion.

64.4.72 assign(Tmatrix3_extended):Tmatrix3_double

Synopsis: Allow assignment of three-dimensional extended precision matrix to three-dimensional double precision matrix.

Declaration: `operator :=(const v: Tmatrix3_extended) : Tmatrix3_double`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with extended precision values wherever a three-dimensional matrix with double precision is expected. Some precision is lost because of the conversion.

64.4.73 assign(Tmatrix3_extended):Tmatrix3_single

Synopsis: Allow assignment of three-dimensional extended precision matrix to three-dimensional single precision matrix.

Declaration: `operator :=(const v: Tmatrix3_extended) : Tmatrix3_single`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with extended precision values wherever a three-dimensional matrix with single precision is expected. Some precision is lost because of the conversion.

64.4.74 assign(Tmatrix3_extended):Tmatrix4_double

Synopsis: Allow assignment of three-dimensional extended precision matrix to four-dimensional double precision matrix.

Declaration: `operator :=(const v: Tmatrix3_extended) : Tmatrix4_double`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with extended precision values wherever a four-dimensional matrix with double precision is expected. Some precision is lost because of the conversion.

64.4.75 assign(Tmatrix3_extended):Tmatrix4_extended

Synopsis: Allow assignment of three-dimensional extended precision matrix to four-dimensional extended precision matrix.

Declaration: `operator :=(const v: Tmatrix3_extended) : Tmatrix4_extended`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with extended precision values wherever a four-dimensional matrix with extended precision is expected.

64.4.76 assign(Tmatrix3_extended):Tmatrix4_single

Synopsis: Allow assignment of three-dimensional extended precision matrix to four-dimensional single precision matrix.

Declaration: `operator :=(const v: Tmatrix3_extended) : Tmatrix4_single`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with extended precision values wherever a four-dimensional matrix with single precision is expected. Some precision is lost because of the conversion.

64.4.77 assign(Tmatrix3_single):Tmatrix2_double

Synopsis: Allow assignment of three-dimensional single precision matrix to two-dimensional double precision matrix.

Declaration: `operator :=(const v: Tmatrix3_single) : Tmatrix2_double`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with single precision values wherever a two-dimensional matrix with double precision is expected. The surplus fields are thrown away.

64.4.78 assign(Tmatrix3_single):Tmatrix2_extended

Synopsis: Allow assignment of three-dimensional single precision matrix to two-dimensional extended precision matrix.

Declaration: `operator :=(const v: Tmatrix3_single) : Tmatrix2_extended`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with single precision values wherever a two-dimensional matrix with extended precision is expected. The surplus fields are thrown away.

64.4.79 **assign(Tmatrix3_single):Tmatrix2_single**

Synopsis: Allow assignment of three-dimensional single precision matrix to two-dimensional single precision matrix.

Declaration: `operator :=(const v: Tmatrix3_single) : Tmatrix2_single`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with single precision values wherever a two-dimensional matrix with single precision is expected. The surplus fields are thrown away.

64.4.80 **assign(Tmatrix3_single):Tmatrix3_double**

Synopsis: Allow assignment of three-dimensional single precision matrix to three-dimensional double precision matrix.

Declaration: `operator :=(const v: Tmatrix3_single) : Tmatrix3_double`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with single precision values wherever a three-dimensional matrix with double precision is expected.

64.4.81 **assign(Tmatrix3_single):Tmatrix3_extended**

Synopsis: Allow assignment of three-dimensional single precision matrix to three-dimensional extended precision matrix.

Declaration: `operator :=(const v: Tmatrix3_single) : Tmatrix3_extended`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with single precision values wherever a three-dimensional matrix with extended precision is expected.

64.4.82 **assign(Tmatrix3_single):Tmatrix4_double**

Synopsis: Allow assignment of three-dimensional single precision matrix to four-dimensional double precision matrix.

Declaration: `operator :=(const v: Tmatrix3_single) : Tmatrix4_double`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with single precision values wherever a four-dimensional matrix with double precision is expected.

64.4.83 assign(Tmatrix3_single):Tmatrix4_extended

Synopsis: Allow assignment of three-dimensional single precision matrix to four-dimensional extended precision matrix.

Declaration: `operator :=(const v: Tmatrix3_single) : Tmatrix4_extended`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with single precision values wherever a four-dimensional matrix with extended precision is expected.

64.4.84 assign(Tmatrix3_single):Tmatrix4_single

Synopsis: Allow assignment of three-dimensional single precision matrix to four-dimensional single precision matrix.

Declaration: `operator :=(const v: Tmatrix3_single) : Tmatrix4_single`

Visibility: default

Description: This operator allows you to use a three-dimensional matrix with single precision values wherever a four-dimensional matrix with single precision is expected.

64.4.85 assign(Tmatrix4_double):Tmatrix2_double

Synopsis: Allow assignment of four-dimensional double precision matrix to two-dimensional double precision matrix.

Declaration: `operator :=(const v: Tmatrix4_double) : Tmatrix2_double`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with double precision values wherever a two-dimensional matrix with double precision is expected. The surplus fields are thrown away.

64.4.86 assign(Tmatrix4_double):Tmatrix2_extended

Synopsis: Allow assignment of four-dimensional double precision matrix to two-dimensional extended precision matrix.

Declaration: `operator :=(const v: Tmatrix4_double) : Tmatrix2_extended`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with double precision values wherever a two-dimensional matrix with extended precision is expected. The surplus fields are thrown away.

64.4.87 assign(Tmatrix4_double):Tmatrix2_single

Synopsis: Allow assignment of four-dimensional double precision matrix to two-dimensional single precision matrix.

Declaration: `operator :=(const v: Tmatrix4_double) : Tmatrix2_single`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with double precision values wherever a two-dimensional matrix with single precision is expected. The surplus fields are thrown away and some precision is lost in the conversion.

64.4.88 assign(Tmatrix4_double):Tmatrix3_double

Synopsis: Allow assignment of four-dimensional double precision matrix to three-dimensional double precision matrix.

Declaration: `operator :=(const v: Tmatrix4_double) : Tmatrix3_double`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with double precision values wherever a three-dimensional matrix with double precision is expected. The surplus fields are thrown away.

64.4.89 assign(Tmatrix4_double):Tmatrix3_extended

Synopsis: Allow assignment of four-dimensional double precision matrix to three-dimensional extended precision matrix.

Declaration: `operator :=(const v: Tmatrix4_double) : Tmatrix3_extended`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with double precision values wherever a three-dimensional matrix with extended precision is expected. The surplus fields are thrown away.

64.4.90 assign(Tmatrix4_double):Tmatrix3_single

Synopsis: Allow assignment of four-dimensional double precision matrix to three-dimensional single precision matrix.

Declaration: `operator :=(const v: Tmatrix4_double) : Tmatrix3_single`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with double precision values wherever a three-dimensional matrix with single precision is expected. The surplus fields are thrown away and some precision is lost because of the conversion.

64.4.91 assign(Tmatrix4_double):Tmatrix4_extended

Synopsis: Allow assignment of four-dimensional double precision matrix to four-dimensional extended precision matrix.

Declaration: `operator :=(const v: Tmatrix4_double) : Tmatrix4_extended`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with double precision values wherever a four-dimensional matrix with extended precision is expected.

64.4.92 assign(Tmatrix4_double):Tmatrix4_single

Synopsis: Allow assignment of four-dimensional single precision matrix to four-dimensional single precision matrix.

Declaration: `operator :=(const v: Tmatrix4_double) : Tmatrix4_single`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with double precision values wherever a four-dimensional matrix with single precision is expected. Some precision is lost because of the conversion.

64.4.93 **assign(Tmatrix4_extended):Tmatrix2_double**

Synopsis: Allow assignment of four-dimensional extended precision matrix to two-dimensional double precision matrix.

Declaration: `operator :=(const v: Tmatrix4_extended) : Tmatrix2_double`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with extended precision values wherever a two-dimensional matrix with double precision is expected. The surplus fields are thrown away and some precision is lost in the conversion.

64.4.94 **assign(Tmatrix4_extended):Tmatrix2_extended**

Synopsis: Allow assignment of four-dimensional extended precision matrix to two-dimensional extended precision matrix.

Declaration: `operator :=(const v: Tmatrix4_extended) : Tmatrix2_extended`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with single precision values wherever a two-dimensional matrix with single precision is expected. The surplus fields are thrown away.

64.4.95 **assign(Tmatrix4_extended):Tmatrix2_single**

Synopsis: Allow assignment of four-dimensional extended precision matrix to two-dimensional single precision matrix.

Declaration: `operator :=(const v: Tmatrix4_extended) : Tmatrix2_single`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with extended precision values wherever a two-dimensional matrix with single precision is expected. The surplus fields are thrown away and some precision is lost in the conversion.

64.4.96 **assign(Tmatrix4_extended):Tmatrix3_double**

Synopsis: Allow assignment of four-dimensional extended precision matrix to three-dimensional double precision matrix.

Declaration: `operator :=(const v: Tmatrix4_extended) : Tmatrix3_double`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with extended precision values wherever a three-dimensional matrix with double precision is expected. The surplus fields are thrown away.

64.4.97 assign(Tmatrix4_extended):Tmatrix3_extended

Synopsis: Allow assignment of four-dimensional extended precision matrix to three-dimensional extended precision matrix.

Declaration: `operator :=(const v: Tmatrix4_extended) : Tmatrix3_extended`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with extended precision values wherever a three-dimensional matrix with double precision is expected. The surplus fields are thrown away.

64.4.98 assign(Tmatrix4_extended):Tmatrix3_single

Synopsis: Allow assignment of four-dimensional extended precision matrix to three-dimensional single precision matrix.

Declaration: `operator :=(const v: Tmatrix4_extended) : Tmatrix3_single`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with extended precision values wherever a three-dimensional matrix with single precision is expected. The surplus fields are thrown away and some precision is lost because of the conversion.

64.4.99 assign(Tmatrix4_extended):Tmatrix4_double

Synopsis: Allow assignment of four-dimensional extended precision matrix to four-dimensional double precision matrix.

Declaration: `operator :=(const v: Tmatrix4_extended) : Tmatrix4_double`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with extended precision values wherever a four-dimensional matrix with double precision is expected.

64.4.100 assign(Tmatrix4_extended):Tmatrix4_single

Synopsis: Allow assignment of four-dimensional extended precision matrix to four-dimensional single precision matrix.

Declaration: `operator :=(const v: Tmatrix4_extended) : Tmatrix4_single`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with extended precision values wherever a four-dimensional matrix with single precision is expected. Some precision is lost because of the conversion.

64.4.101 assign(Tmatrix4_single):Tmatrix2_double

Synopsis: Allow assignment of four-dimensional single precision matrix to two-dimensional double precision matrix.

Declaration: `operator :=(const v: Tmatrix4_single) : Tmatrix2_double`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with single precision values wherever a two-dimensional matrix with double precision is expected. The surplus fields are thrown away.

64.4.102 **assign(Tmatrix4_single):Tmatrix2_extended**

Synopsis: Allow assignment of four-dimensional single precision matrix to two-dimensional extended precision matrix.

Declaration: `operator :=(const v: Tmatrix4_single) : Tmatrix2_extended`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with single precision values wherever a two-dimensional matrix with extended precision is expected. The surplus fields are thrown away.

64.4.103 **assign(Tmatrix4_single):Tmatrix2_single**

Synopsis: Allow assignment of four-dimensional single precision matrix to two-dimensional single precision matrix.

Declaration: `operator :=(const v: Tmatrix4_single) : Tmatrix2_single`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with single precision values wherever a two-dimensional matrix with single precision is expected. The surplus fields are thrown away.

64.4.104 **assign(Tmatrix4_single):Tmatrix3_double**

Synopsis: Allow assignment of four-dimensional single precision matrix to three-dimensional double precision matrix.

Declaration: `operator :=(const v: Tmatrix4_single) : Tmatrix3_double`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with single precision values wherever a three-dimensional matrix with double precision is expected. The surplus fields are thrown away.

64.4.105 **assign(Tmatrix4_single):Tmatrix3_extended**

Synopsis: Allow assignment of four-dimensional single precision matrix to three-dimensional extended precision matrix.

Declaration: `operator :=(const v: Tmatrix4_single) : Tmatrix3_extended`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with single precision values wherever a three-dimensional matrix with extended precision is expected. The surplus fields are thrown away.

64.4.106 assign(Tmatrix4_single):Tmatrix3_single

Synopsis: Allow assignment of four-dimensional single precision matrix to three-dimensional single precision matrix.

Declaration: `operator :=(const v: Tmatrix4_single) : Tmatrix3_single`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with single precision values wherever a three-dimensional matrix with single precision is expected. The surplus fields are thrown away.

64.4.107 assign(Tmatrix4_single):Tmatrix4_double

Synopsis: Allow assignment of four-dimensional single precision matrix to four-dimensional double precision matrix.

Declaration: `operator :=(const v: Tmatrix4_single) : Tmatrix4_double`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with single precision values wherever a four-dimensional matrix with double precision is expected.

64.4.108 assign(Tmatrix4_single):Tmatrix4_extended

Synopsis: Allow assignment of four-dimensional single precision matrix to four-dimensional extended precision matrix.

Declaration: `operator :=(const v: Tmatrix4_single) : Tmatrix4_extended`

Visibility: default

Description: This operator allows you to use a four-dimensional matrix with single precision values wherever a four-dimensional matrix with extended precision is expected.

64.4.109 assign(Tvector2_double):Tvector2_extended

Synopsis: Allow assignment of double precision vector to extended precision vector.

Declaration: `operator :=(const v: Tvector2_double) : Tvector2_extended`

Visibility: default

Description: This operator allows you to use a vector with double precision values wherever an extended precision vector is expected.

64.4.110 assign(Tvector2_double):Tvector2_single

Synopsis: Allow assignment of double precision vector to single precision vector.

Declaration: `operator :=(const v: Tvector2_double) : Tvector2_single`

Visibility: default

Description: This operator allows you to use a vector with double precision values wherever a single precision vector is expected, at the cost of loosing some precision.

64.4.111 assign(Tvector2_double):Tvector3_double

Synopsis: Allow assignment of two-dimensional double precision vector to three-dimensional double precision vector.

Declaration: `operator :=(const v: Tvector2_double) : Tvector3_double`

Visibility: default

Description: This operator allows you to use a two-dimensional vector with double precision values wherever a three-dimensional vector with double precision is expected. The third dimension is set to 0.0.

64.4.112 assign(Tvector2_double):Tvector3_extended

Synopsis: Allow assignment of two-dimensional double precision vector to three-dimensional extended precision vector.

Declaration: `operator :=(const v: Tvector2_double) : Tvector3_extended`

Visibility: default

Description: This operator allows you to use a two-dimensional vector with double precision values wherever a three-dimensional vector with extended precision is expected. The third dimension is set to 0.0.

64.4.113 assign(Tvector2_double):Tvector3_single

Synopsis: Allow assignment of two-dimensional double precision vector to three-dimensional single precision vector.

Declaration: `operator :=(const v: Tvector2_double) : Tvector3_single`

Visibility: default

Description: This operator allows you to use a two-dimensional vector with double precision values wherever a three-dimensional vector with single precision is expected. Some accuracy is lost because of the conversion and the third dimension is set to 0.0.

64.4.114 assign(Tvector2_double):Tvector4_double

Synopsis: Allow assignment of two-dimensional double precision vector to four-dimensional double precision vector.

Declaration: `operator :=(const v: Tvector2_double) : Tvector4_double`

Visibility: default

Description: This operator allows you to use a two-dimensional vector with double precision values wherever a four-dimensional vector with double precision is expected. The third and fourth dimensions are set to 0.0.

64.4.115 assign(Tvector2_double):Tvector4_extended

Synopsis: Allow assignment of two-dimensional double precision vector to four-dimensional extended precision vector.

Declaration: `operator :=(const v: Tvector2_double) : Tvector4_extended`

Visibility: default

Description: This operator allows you to use a two-dimensional vector with single precision values wherever a four-dimensional vector with extended precision is expected. The third and fourth dimensions are set to 0.0.

64.4.116 `assign(Tvector2_double):Tvector4_single`

Synopsis: Allow assignment of two-dimensional double precision vector to four-dimensional single precision vector.

Declaration: `operator :=(const v: Tvector2_double) : Tvector4_single`

Visibility: default

Description: This operator allows you to use a two-dimensional vector with double precision values wherever a four-dimensional vector with single precision is expected. Some accuracy is lost because of the conversion and the third and fourth dimensions are set to 0.0.

64.4.117 `assign(Tvector2_extended):Tvector2_double`

Synopsis: Allow assignment of extended precision vector to double precision vector.

Declaration: `operator :=(const v: Tvector2_extended) : Tvector2_double`

Visibility: default

Description: This operator allows you to use a vector with extended precision values wherever a double precision vector is expected, at the cost of loosing some precision.

64.4.118 `assign(Tvector2_extended):Tvector2_single`

Synopsis: Allow assignment of extended precision vector to single precision vector.

Declaration: `operator :=(const v: Tvector2_extended) : Tvector2_single`

Visibility: default

Description: This operator allows you to use a vector with extended precision values wherever a single precision vector is expected, at the cost of loosing some precision.

64.4.119 `assign(Tvector2_extended):Tvector3_double`

Synopsis: Allow assignment of two-dimensional extended precision vector to three-dimensional double precision vector.

Declaration: `operator :=(const v: Tvector2_extended) : Tvector3_double`

Visibility: default

Description: This operator allows you to use a two-dimensional vector with extended precision values wherever a three-dimensional vector with double precision is expected. Some accuracy is lost because of the conversion and the third dimension is set to 0.0.

64.4.120 assign(Tvector2_extended):Tvector3_extended

Synopsis: Allow assignment of two-dimensional extended precision vector to three-dimensional extended precision vector.

Declaration: `operator :=(const v: Tvector2_extended) : Tvector3_extended`

Visibility: default

Description: This operator allows you to use a two-dimensional vector with extended precision values wherever a three-dimensional vector with extended precision is expected. The third dimension is set to 0.0.

64.4.121 assign(Tvector2_extended):Tvector3_single

Synopsis: Allow assignment of two-dimensional extended precision vector to three-dimensional single precision vector.

Declaration: `operator :=(const v: Tvector2_extended) : Tvector3_single`

Visibility: default

Description: This operator allows you to use a two-dimensional vector with extended precision values wherever a three-dimensional vector with single precision is expected. Some accuracy is lost because of the conversion and the third dimension is set to 0.0.

64.4.122 assign(Tvector2_extended):Tvector4_double

Synopsis: Allow assignment of two-dimensional extended precision vector to four-dimensional double precision vector.

Declaration: `operator :=(const v: Tvector2_extended) : Tvector4_double`

Visibility: default

Description: This operator allows you to use a two-dimensional vector with extended precision values wherever a four-dimensional vector with double precision is expected. Some accuracy is lost because of the conversion and the third and fourth dimensions are set to 0.0.

64.4.123 assign(Tvector2_extended):Tvector4_extended

Synopsis: Allow assignment of two-dimensional extended precision vector to four-dimensional extended precision vector.

Declaration: `operator :=(const v: Tvector2_extended) : Tvector4_extended`

Visibility: default

Description: This operator allows you to use a two-dimensional vector with extended precision values wherever a four-dimensional vector with extended precision is expected. The third and fourth dimensions are set to 0.0.

64.4.124 assign(Tvector2_extended):Tvector4_single

Synopsis: Allow assignment of two-dimensional extended precision vector to four-dimensional single precision vector.

Declaration: `operator :=(const v: Tvector2_extended) : Tvector4_single`

Visibility: default

Description: This operator allows you to use a two-dimensional vector with extended precision values wherever a four-dimensional vector with single precision is expected. Some accuracy is lost because of the conversion and the third and fourth dimensions are set to 0.0.

64.4.125 assign(Tvector2_single):Tvector2_double

Synopsis: Allow assignment of single precision vector to double precision vector.

Declaration: `operator :=(const v: Tvector2_single) : Tvector2_double`

Visibility: default

Description: This operator allows you to use a vector with single precision values wherever a double precision vector is expected.

64.4.126 assign(Tvector2_single):Tvector2_extended

Synopsis: Allow assignment of single precision vector to extended precision vector.

Declaration: `operator :=(const v: Tvector2_single) : Tvector2_extended`

Visibility: default

Description: This operator allows you to use a vector with single precision values wherever an extended precision vector is expected.

64.4.127 assign(Tvector2_single):Tvector3_double

Synopsis: Allow assignment of two-dimensional single precision vector to three-dimensional double precision vector.

Declaration: `operator :=(const v: Tvector2_single) : Tvector3_double`

Visibility: default

Description: This operator allows you to use a two-dimensional vector with single precision values wherever a three-dimensional vector with double precision is expected. The third dimension is set to 0.0.

64.4.128 assign(Tvector2_single):Tvector3_extended

Synopsis: Allow assignment of two-dimensional single precision vector to three-dimensional extended precision vector.

Declaration: `operator :=(const v: Tvector2_single) : Tvector3_extended`

Visibility: default

Description: This operator allows you to use a two-dimensional vector with single precision values wherever a three-dimensional vector with extended precision is expected. The third dimension is set to 0.0.

64.4.129 assign(Tvector2_single):Tvector3_single

Synopsis: Allow assignment of two-dimensional single precision vector to three-dimensional single precision vector.

Declaration: `operator :=(const v: Tvector2_single) : Tvector3_single`

Visibility: default

Description: This operator allows you to use a two-dimensional vector with single precision values wherever a three-dimensional vector with single precision is expected. The third dimension is set to 0.0.

64.4.130 assign(Tvector2_single):Tvector4_double

Synopsis: Allow assignment of two-dimensional single precision vector to four-dimensional double precision vector.

Declaration: `operator :=(const v: Tvector2_single) : Tvector4_double`

Visibility: default

Description: This operator allows you to use a two-dimensional vector with single precision values wherever a four-dimensional vector with double precision is expected. The third and fourth dimensions are set to 0.0.

64.4.131 assign(Tvector2_single):Tvector4_extended

Synopsis: Allow assignment of two-dimensional single precision vector to four-dimensional extended precision vector.

Declaration: `operator :=(const v: Tvector2_single) : Tvector4_extended`

Visibility: default

Description: This operator allows you to use a two-dimensional vector with single precision values wherever a four-dimensional vector with extended precision is expected. The third and fourth dimensions are set to 0.0.

64.4.132 assign(Tvector2_single):Tvector4_single

Synopsis: Allow assignment of two-dimensional single precision vector to four-dimensional single precision vector.

Declaration: `operator :=(const v: Tvector2_single) : Tvector4_single`

Visibility: default

Description: This operator allows you to use a two-dimensional vector with single precision values wherever a four-dimensional vector with single precision is expected. The third and fourth dimensions are set to 0.0.

64.4.133 assign(Tvector3_double):Tvector2_double

Synopsis: Allow assignment of three-dimensional double precision vector to two-dimensional double precision vector.

Declaration: `operator :=(const v: Tvector3_double) : Tvector2_double`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with double precision values wherever a two-dimensional vector with double precision is expected. The third dimension is thrown away.

64.4.134 assign(Tvector3_double):Tvector2_extended

Synopsis: Allow assignment of three-dimensional double precision vector to two-dimensional extended precision vector.

Declaration: `operator :=(const v: Tvector3_double) : Tvector2_extended`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with double precision values wherever a two-dimensional vector with extended precision is expected. The third dimension is thrown away.

64.4.135 assign(Tvector3_double):Tvector2_single

Synopsis: Allow assignment of three-dimensional double precision vector to two-dimensional single precision vector.

Declaration: `operator :=(const v: Tvector3_double) : Tvector2_single`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with double precision values wherever a two-dimensional vector with single precision is expected. The third dimension is thrown away and some precision is lost because of the conversion.

64.4.136 assign(Tvector3_double):Tvector3_extended

Synopsis: Allow assignment of three-dimensional double precision vector to three-dimensional extended precision vector.

Declaration: `operator :=(const v: Tvector3_double) : Tvector3_extended`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with double precision values wherever a three-dimensional vector with extended precision is expected.

64.4.137 assign(Tvector3_double):Tvector3_single

Synopsis: Allow assignment of three-dimensional double precision vector to three-dimensional single precision vector.

Declaration: `operator :=(const v: Tvector3_double) : Tvector3_single`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with double precision values wherever a three-dimensional vector with single precision is expected. Some precision is lost because of the conversion.

64.4.138 **assign(Tvector3_double):Tvector4_double**

Synopsis: Allow assignment of three-dimensional double precision vector to four-dimensional double precision vector.

Declaration: `operator :=(const v: Tvector3_double) : Tvector4_double`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with double precision values wherever a four-dimensional vector with double precision is expected. The fourth dimension is set to 0.

64.4.139 **assign(Tvector3_double):Tvector4_extended**

Synopsis: Allow assignment of three-dimensional double precision vector to four-dimensional extended precision vector.

Declaration: `operator :=(const v: Tvector3_double) : Tvector4_extended`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with double precision values wherever a four-dimensional vector with extended precision is expected. The fourth dimension is set to 0.

64.4.140 **assign(Tvector3_double):Tvector4_single**

Synopsis: Allow assignment of three-dimensional double precision vector to four-dimensional single precision vector.

Declaration: `operator :=(const v: Tvector3_double) : Tvector4_single`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with single precision values wherever a four-dimensional vector with double precision is expected. The fourth dimension is set to 0 and some precision is lost because of the conversion.

64.4.141 **assign(Tvector3_extended):Tvector2_double**

Synopsis: Allow assignment of three-dimensional extended precision vector to two-dimensional double precision vector.

Declaration: `operator :=(const v: Tvector3_extended) : Tvector2_double`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with extended precision values wherever a two-dimensional vector with double precision is expected. The third dimension is thrown away and some precision is lost because of the conversion.

64.4.142 assign(Tvector3_extended):Tvector2_extended

Synopsis: Allow assignment of three-dimensional extended precision vector to two-dimensional extended precision vector.

Declaration: `operator :=(const v: Tvector3_extended) : Tvector2_extended`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with extended precision values wherever a two-dimensional vector with extended precision is expected. The third dimension is thrown away.

64.4.143 assign(Tvector3_extended):Tvector2_single

Synopsis: Allow assignment of three-dimensional extended precision vector to two-dimensional single precision vector.

Declaration: `operator :=(const v: Tvector3_extended) : Tvector2_single`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with extended precision values wherever a two-dimensional vector with single precision is expected. The third dimension is thrown away and some precision is lost because of the conversion.

64.4.144 assign(Tvector3_extended):Tvector3_double

Synopsis: Allow assignment of three-dimensional extended precision vector to three-dimensional double precision vector.

Declaration: `operator :=(const v: Tvector3_extended) : Tvector3_double`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with extended precision values wherever a three-dimensional vector with double precision is expected. Some precision is lost because of the conversion.

64.4.145 assign(Tvector3_extended):Tvector3_single

Synopsis: Allow assignment of three-dimensional single precision vector to three-dimensional double precision vector.

Declaration: `operator :=(const v: Tvector3_extended) : Tvector3_single`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with single precision values wherever a three-dimensional vector with double precision is expected. Some precision is lost because of the conversion.

64.4.146 assign(Tvector3_extended):Tvector4_double

Synopsis: Allow assignment of three-dimensional extended precision vector to four-dimensional double precision vector.

Declaration: `operator :=(const v: Tvector3_extended) : Tvector4_double`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with extended precision values wherever a four-dimensional vector with double precision is expected. The fourth dimension is set to 0 and some accuracy is lost because of the conversion.

64.4.147 assign(Tvector3_extended):Tvector4_extended

Synopsis: Allow assignment of three-dimensional extended precision vector to four-dimensional extended precision vector.

Declaration: `operator :=(const v: Tvector3_extended) : Tvector4_extended`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with extended precision values wherever a four-dimensional vector with extended precision is expected. The fourth dimension is set to 0.

64.4.148 assign(Tvector3_extended):Tvector4_single

Synopsis: Allow assignment of three-dimensional extended precision vector to four-dimensional single precision vector.

Declaration: `operator :=(const v: Tvector3_extended) : Tvector4_single`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with extended precision values wherever a four-dimensional vector with single precision is expected. The fourth dimension is set to 0 and some accuracy is lost because of the conversion.

64.4.149 assign(Tvector3_single):Tvector2_double

Synopsis: Allow assignment of three-dimensional single precision vector to two-dimensional double precision vector.

Declaration: `operator :=(const v: Tvector3_single) : Tvector2_double`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with single precision values wherever a two-dimensional vector with double precision is expected. The third dimension is thrown away.

64.4.150 assign(Tvector3_single):Tvector2_extended

Synopsis: Allow assignment of three-dimensional single precision vector to two-dimensional extended precision vector.

Declaration: `operator :=(const v: Tvector3_single) : Tvector2_extended`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with single precision values wherever a two-dimensional vector with extended precision is expected. The third dimension is thrown away.

64.4.151 **assign(Tvector3_single):Tvector2_single**

Synopsis: Allow assignment of three-dimensional single precision vector to two-dimensional single precision vector.

Declaration: `operator :=(const v: Tvector3_single) : Tvector2_single`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with single precision values wherever a two-dimensional vector with single precision is expected. The third dimension is thrown away.

64.4.152 **assign(Tvector3_single):Tvector3_double**

Synopsis: Allow assignment of three-dimensional single precision vector to three-dimensional double precision vector.

Declaration: `operator :=(const v: Tvector3_single) : Tvector3_double`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with single precision values wherever a three-dimensional vector with double precision is expected.

64.4.153 **assign(Tvector3_single):Tvector3_extended**

Synopsis: Allow assignment of three-dimensional single precision vector to three-dimensional extended precision vector.

Declaration: `operator :=(const v: Tvector3_single) : Tvector3_extended`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with single precision values wherever a three-dimensional vector with extended precision is expected.

64.4.154 **assign(Tvector3_single):Tvector4_double**

Synopsis: Allow assignment of three-dimensional single precision vector to four-dimensional double precision vector.

Declaration: `operator :=(const v: Tvector3_single) : Tvector4_double`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with single precision values wherever a four-dimensional vector with double precision is expected. The fourth dimension is set to 0.

64.4.155 assign(Tvector3_single):Tvector4_extended

Synopsis: Allow assignment of three-dimensional single precision vector to four-dimensional extended precision vector.

Declaration: `operator :=(const v: Tvector3_single) : Tvector4_extended`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with single precision values wherever a four-dimensional vector with extended precision is expected. The fourth dimension is set to 0.

64.4.156 assign(Tvector3_single):Tvector4_single

Synopsis: Allow assignment of three-dimensional single precision vector to four-dimensional single precision vector.

Declaration: `operator :=(const v: Tvector3_single) : Tvector4_single`

Visibility: default

Description: This operator allows you to use a three-dimensional vector with single precision values wherever a four-dimensional vector with single precision is expected. The fourth dimension is set to 0.

64.4.157 assign(Tvector4_double):Tvector2_double

Synopsis: Allow assignment of four-dimensional double precision vector to two-dimensional double precision vector.

Declaration: `operator :=(const v: Tvector4_double) : Tvector2_double`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with double precision values wherever a two-dimensional vector with double precision is expected. The third and fourth dimensions are thrown away.

64.4.158 assign(Tvector4_double):Tvector2_extended

Synopsis: Allow assignment of four-dimensional double precision vector to two-dimensional extended precision vector.

Declaration: `operator :=(const v: Tvector4_double) : Tvector2_extended`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with double precision values wherever a two-dimensional vector with extended precision is expected. The third and fourth dimensions are thrown away.

64.4.159 assign(Tvector4_double):Tvector2_single

Synopsis: Allow assignment of four-dimensional double precision vector to two-dimensional single precision vector.

Declaration: `operator :=(const v: Tvector4_double) : Tvector2_single`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with double precision values wherever a two-dimensional vector with single precision is expected. The third and fourth dimensions are thrown away and some accuracy is lost because of the conversion.

64.4.160 **assign(Tvector4_double):Tvector3_double**

Synopsis: Allow assignment of four-dimensional double precision vector to three-dimensional double precision vector.

Declaration: `operator :=(const v: Tvector4_double) : Tvector3_double`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with double precision values wherever a three-dimensional vector with double precision is expected. The fourth dimension is thrown away.

64.4.161 **assign(Tvector4_double):Tvector3_extended**

Synopsis: Allow assignment of four-dimensional double precision vector to three-dimensional extended precision vector.

Declaration: `operator :=(const v: Tvector4_double) : Tvector3_extended`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with double precision values wherever a three-dimensional vector with extended precision is expected. The fourth dimension is thrown away.

64.4.162 **assign(Tvector4_double):Tvector3_single**

Synopsis: Allow assignment of four-dimensional double precision vector to three-dimensional single precision vector.

Declaration: `operator :=(const v: Tvector4_double) : Tvector3_single`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with double precision values wherever a three-dimensional vector with single precision is expected. The fourth dimension is thrown away and some accuracy is lost because of the conversion.

64.4.163 **assign(Tvector4_double):Tvector4_extended**

Synopsis: Allow assignment of four-dimensional single precision vector to four-dimensional extended precision vector.

Declaration: `operator :=(const v: Tvector4_double) : Tvector4_extended`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with double precision values wherever a four-dimensional vector with extended precision is expected.

64.4.164 assign(Tvector4_double):Tvector4_single

Synopsis: Allow assignment of four-dimensional double precision vector to four-dimensional single precision vector.

Declaration: `operator :=(const v: Tvector4_double) : Tvector4_single`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with double precision values wherever a four-dimensional vector with single precision is expected. Some accuracy is lost because of the conversion.

64.4.165 assign(Tvector4_extended):Tvector2_double

Synopsis: Allow assignment of four-dimensional extended precision vector to two-dimensional double precision vector.

Declaration: `operator :=(const v: Tvector4_extended) : Tvector2_double`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with extended precision values wherever a two-dimensional vector with double precision is expected. The third and fourth dimensions are thrown away and some accuracy is lost because of the conversion.

64.4.166 assign(Tvector4_extended):Tvector2_extended

Synopsis: Allow assignment of four-dimensional extended precision vector to two-dimensional extended precision vector.

Declaration: `operator :=(const v: Tvector4_extended) : Tvector2_extended`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with extended precision values wherever a two-dimensional vector with extended precision is expected. The third and fourth dimensions are thrown away.

64.4.167 assign(Tvector4_extended):Tvector2_single

Synopsis: Allow assignment of four-dimensional extended precision vector to two-dimensional single precision vector.

Declaration: `operator :=(const v: Tvector4_extended) : Tvector2_single`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with extended precision values wherever a two-dimensional vector with single precision is expected. The third and fourth dimensions are thrown away and some accuracy is lost because of the conversion.

64.4.168 assign(Tvector4_extended):Tvector3_double

Synopsis: Allow assignment of four-dimensional extended precision vector to three-dimensional double precision vector.

Declaration: `operator :=(const v: Tvector4_extended) : Tvector3_double`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with extended precision values wherever a three-dimensional vector with double precision is expected. The fourth dimension is thrown away and some accuracy is lost because of the conversion.

64.4.169 assign(Tvector4_extended):Tvector3_extended

Synopsis: Allow assignment of four-dimensional extended precision vector to three-dimensional extended precision vector.

Declaration: `operator :=(const v: Tvector4_extended) : Tvector3_extended`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with extended precision values wherever a three-dimensional vector with extended precision is expected. The fourth dimensions are thrown away.

64.4.170 assign(Tvector4_extended):Tvector3_single

Synopsis: Allow assignment of four-dimensional extended precision vector to three-dimensional single precision vector.

Declaration: `operator :=(const v: Tvector4_extended) : Tvector3_single`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with extended precision values wherever a three-dimensional vector with single precision is expected. The fourth dimension is thrown away and some accuracy is lost because of the conversion.

64.4.171 assign(Tvector4_extended):Tvector4_double

Synopsis: Allow assignment of four-dimensional single precision vector to four-dimensional double precision vector.

Declaration: `operator :=(const v: Tvector4_extended) : Tvector4_double`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with extended precision values wherever a four-dimensional vector with double precision is expected. Some accuracy is lost because of the conversion.

64.4.172 assign(Tvector4_extended):Tvector4_single

Synopsis: Allow assignment of four-dimensional extended precision vector to four-dimensional single precision vector.

Declaration: `operator :=(const v: Tvector4_extended) : Tvector4_single`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with extended precision values wherever a four-dimensional vector with single precision is expected. Some accuracy is lost because of the conversion.

64.4.173 assign(Tvector4_single):Tvector2_double

Synopsis: Allow assignment of four-dimensional single precision vector to two-dimensional double precision vector.

Declaration: `operator :=(const v: Tvector4_single) : Tvector2_double`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with single precision values wherever a two-dimensional vector with double precision is expected. The third and fourth dimensions are thrown away.

64.4.174 assign(Tvector4_single):Tvector2_extended

Synopsis: Allow assignment of four-dimensional single precision vector to two-dimensional extended precision vector.

Declaration: `operator :=(const v: Tvector4_single) : Tvector2_extended`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with single precision values wherever a two-dimensional vector with extended precision is expected. The third and fourth dimensions are thrown away.

64.4.175 assign(Tvector4_single):Tvector2_single

Synopsis: Allow assignment of four-dimensional single precision vector to two-dimensional single precision vector.

Declaration: `operator :=(const v: Tvector4_single) : Tvector2_single`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with single precision values wherever a two-dimensional vector with single precision is expected. The third and fourth dimensions are thrown away.

64.4.176 assign(Tvector4_single):Tvector3_double

Synopsis: Allow assignment of four-dimensional single precision vector to three-dimensional double precision vector.

Declaration: `operator :=(const v: Tvector4_single) : Tvector3_double`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with single precision values wherever a three-dimensional vector with double precision is expected. The fourth dimension is thrown away.

64.4.177 assign(Tvector4_single):Tvector3_extended

Synopsis: Allow assignment of four-dimensional single precision vector to three-dimensional extended precision vector.

Declaration: `operator :=(const v: Tvector4_single) : Tvector3_extended`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with single precision values wherever a three-dimensional vector with extended precision is expected. The fourth dimension is thrown away.

64.4.178 assign(Tvector4_single):Tvector3_single

Synopsis: Allow assignment of four-dimensional single precision vector to three-dimensional single precision vector.

Declaration: `operator :=(const v: Tvector4_single) : Tvector3_single`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with single precision values wherever a three-dimensional vector with single precision is expected. The fourth dimension is thrown away.

64.4.179 assign(Tvector4_single):Tvector4_double

Synopsis: Allow assignment of four-dimensional single precision vector to four-dimensional double precision vector.

Declaration: `operator :=(const v: Tvector4_single) : Tvector4_double`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with single precision values wherever a four-dimensional vector with double precision is expected.

64.4.180 assign(Tvector4_single):Tvector4_extended

Synopsis: Allow assignment of four-dimensional single precision vector to four-dimensional extended precision vector.

Declaration: `operator :=(const v: Tvector4_single) : Tvector4_extended`

Visibility: default

Description: This operator allows you to use a four-dimensional vector with single precision values wherever a four-dimensional vector with extended precision is expected.

64.4.181 divide(Tmatrix2_double,Double):Tmatrix2_double

Synopsis: Divide a two-dimensional single precision matrix by a scalar.

Declaration: `operator / (const m: Tmatrix2_double; const x: Double) : Tmatrix2_double`

Visibility: default

Description: This operator allows you to divide a matrix by a scalar. All elements in the matrix are divided by the scalar, the result is returned as a new matrix.

64.4.182 divide(Tmatrix2_extended,extended):Tmatrix2_extended

Synopsis: Divide a two-dimensional single precision matrix by a scalar.

Declaration: `operator / (const m: Tmatrix2_extended; const x: extended) : Tmatrix2_extended`

Visibility: default

Description: This operator allows you to divide a matrix by a scalar. All elements in the matrix are divided by the scalar, the result is returned as a new matrix.

64.4.183 divide(Tmatrix2_single,single):Tmatrix2_single

Synopsis: Divide a two-dimensional single precision matrix by a scalar.

Declaration: `operator / (const m: Tmatrix2_single; const x: single) : Tmatrix2_single`

Visibility: default

Description: This operator allows you to divide a matrix by a scalar. All elements in the matrix are divided by the scalar, the result is returned as a new matrix.

64.4.184 divide(Tmatrix3_double,Double):Tmatrix3_double

Synopsis: Divide a two-dimensional single precision matrix by a scalar.

Declaration: `operator / (const m: Tmatrix3_double; const x: Double) : Tmatrix3_double`

Visibility: default

Description: This operator allows you to divide a matrix by a scalar. All elements in the matrix are divided by the scalar, the result is returned as a new matrix.

64.4.185 divide(Tmatrix3_extended,extended):Tmatrix3_extended

Synopsis: Divide a two-dimensional single precision matrix by a scalar.

Declaration: `operator / (const m: Tmatrix3_extended; const x: extended) : Tmatrix3_extended`

Visibility: default

Description: This operator allows you to divide a matrix by a scalar. All elements in the matrix are divided by the scalar, the result is returned as a new matrix.

64.4.186 divide(Tmatrix3_single,single):Tmatrix3_single

Synopsis: Divide a two-dimensional single precision matrix by a scalar.

Declaration: `operator / (const m: Tmatrix3_single; const x: single) : Tmatrix3_single`

Visibility: default

Description: This operator allows you to divide a matrix by a scalar. All elements in the matrix are divided by the scalar, the result is returned as a new matrix.

64.4.187 divide(Tmatrix4_double,Double):Tmatrix4_double

Synopsis: Divide a two-dimensional single precision matrix by a scalar.

Declaration: `operator / (const m: Tmatrix4_double; const x: Double) : Tmatrix4_double`

Visibility: default

Description: This operator allows you to divide a matrix by a scalar. All elements in the matrix are divided by the scalar, the result is returned as a new matrix.

64.4.188 divide(Tmatrix4_extended,extended):Tmatrix4_extended

Synopsis: Divide a two-dimensional single precision matrix by a scalar.

Declaration: `operator / (const m: Tmatrix4_extended; const x: extended) : Tmatrix4_extended`

Visibility: default

Description: This operator allows you to divide a matrix by a scalar. All elements in the matrix are divided by the scalar, the result is returned as a new matrix.

64.4.189 divide(Tmatrix4_single,single):Tmatrix4_single

Synopsis: Divide a two-dimensional single precision matrix by a scalar.

Declaration: `operator / (const m: Tmatrix4_single; const x: single) : Tmatrix4_single`

Visibility: default

Description: This operator allows you to divide a matrix by a scalar. All elements in the matrix are divided by the scalar, the result is returned as a new matrix.

64.4.190 divide(Tvector2_double,Double):Tvector2_double

Synopsis: Divide a two-dimensional double precision vector by a scalar.

Declaration: `operator / (const x: Tvector2_double; y: Double) : Tvector2_double`

Visibility: default

Description: This operator allows you to divide a vector by a scalar value. Each vector element is divided by the scalar value; the result is returned as a new vector.

64.4.191 divide(Tvector2_extended,extended):Tvector2_extended

Synopsis: Divide a two-dimensional extended precision vector by a scalar.

Declaration: `operator / (const x: Tvector2_extended; y: extended) : Tvector2_extended`

Visibility: default

Description: This operator allows you to divide a vector by a scalar value. Each vector element is divided by the scalar value; the result is returned as a new vector.

64.4.192 divide(Tvector2_single,single):Tvector2_single

Synopsis: Divide a two-dimensional single precision vector by a scalar.

Declaration: `operator / (const x: Tvector2_single; y: single) : Tvector2_single`

Visibility: default

Description: This operator allows you to divide a vector by a scalar value. Each vector element is divided by the scalar value; the result is returned as a new vector.

64.4.193 divide(Tvector3_double,Double):Tvector3_double

Synopsis: Divide a three-dimensional double precision vector by a scalar.

Declaration: `operator / (const x: Tvector3_double; y: Double) : Tvector3_double`

Visibility: default

Description: This operator allows you to divide a vector by a scalar value. Each vector element is divided by the scalar value; the result is returned as a new vector.

64.4.194 divide(Tvector3_extended,extended):Tvector3_extended

Synopsis: Divide a three-dimensional extended precision vector by a scalar.

Declaration: `operator / (const x: Tvector3_extended; y: extended) : Tvector3_extended`

Visibility: default

Description: This operator allows you to divide a vector by a scalar value. Each vector element is divided by the scalar value; the result is returned as a new vector.

64.4.195 divide(Tvector3_single,single):Tvector3_single

Synopsis: Divide a three-dimensional single precision vector by a scalar.

Declaration: `operator / (const x: Tvector3_single; y: single) : Tvector3_single`

Visibility: default

Description: This operator allows you to divide a vector by a scalar value. Each vector element is divided by the scalar value; the result is returned as a new vector.

64.4.196 divide(Tvector4_double,Double):Tvector4_double

Synopsis: Divide a four-dimensional double precision vector by a scalar.

Declaration: `operator / (const x: Tvector4_double; y: Double) : Tvector4_double`

Visibility: default

Description: This operator allows you to divide a vector by a scalar value. Each vector element is divided by the scalar value; the result is returned as a new vector.

64.4.197 divide(Tvector4_extended,extended):Tvector4_extended

Synopsis: Divide a four-dimensional extended precision vector by a scalar.

Declaration: `operator / (const x: Tvector4_extended; y: extended) : Tvector4_extended`

Visibility: default

Description: This operator allows you to divide a vector by a scalar value. Each vector element is divided by the scalar value; the result is returned as a new vector.

64.4.198 divide(Tvector4_single,single):Tvector4_single

Synopsis: Divide a four-dimensional single precision vector by a scalar.

Declaration: `operator / (const x: Tvector4_single; y: single) : Tvector4_single`

Visibility: default

Description: This operator allows you to divide a vector by a scalar value. Each vector element is divided by the scalar value; the result is returned as a new vector.

64.4.199 multiply(Tmatrix2_double,Double):Tmatrix2_double

Synopsis: Multiply a two-dimensional double precision matrix by a scalar.

Declaration: `operator * (const m: Tmatrix2_double; const x: Double) : Tmatrix2_double`

Visibility: default

Description: This operator allows you to multiply a matrix with a scalar. All elements in the matrix are multiplied by the scalar, the result is returned as a new matrix.

64.4.200 multiply(Tmatrix2_double,Tmatrix2_double):Tmatrix2_double

Synopsis: Give product of two two-dimensional double precision matrices.

Declaration: `operator * (const m1: Tmatrix2_double; const m2: Tmatrix2_double)
: Tmatrix2_double`

Visibility: default

Description: This operator allows you to multiply two two-dimensional single precision matrices. A new matrix is returned which is the product of both matrices. The product is calculated using the well known matrix multiplication algorithm.

64.4.201 multiply(Tmatrix2_double,Tvector2_double):Tvector2_double

Synopsis: Give product of a two-dimensional double precision matrix and vector.

Declaration: `operator *(const m: Tmatrix2_double; const v: Tvector2_double)
: Tvector2_double`

Visibility: default

Description: This operator allows you to multiply a two-dimensional double precision matrices with a two dimensional double precision vector. A new vector is returned which is the product of the matrix and the vector. The product is calculated using the well known matrix-vector multiplication algorithm.

64.4.202 multiply(Tmatrix2_extended,extended):Tmatrix2_extended

Synopsis: Multiply a two-dimensional extended precision matrix by a scalar.

Declaration: `operator *(const m: Tmatrix2_extended; const x: extended)
: Tmatrix2_extended`

Visibility: default

Description: This operator allows you to multiply a matrix with a scalar. All elements in the matrix are multiplied by the scalar, the result is returned as a new matrix.

64.4.203 multiply(Tmatrix2_extended,Tmatrix2_extended):Tmatrix2_extended

Synopsis: Give product of two two-dimensional extended precision matrices.

Declaration: `operator *(const m1: Tmatrix2_extended; const m2: Tmatrix2_extended)
: Tmatrix2_extended`

Visibility: default

Description: This operator allows you to multiply two two-dimensional single precision matrices. A new matrix is returned which is the product of both matrices. The product is calculated using the well known matrix multiplication algorithm.

64.4.204 multiply(Tmatrix2_extended,Tvector2_extended):Tvector2_extended

Synopsis: Give product of a two-dimensional extended precision matrix and vector.

Declaration: `operator *(const m: Tmatrix2_extended; const v: Tvector2_extended)
: Tvector2_extended`

Visibility: default

Description: This operator allows you to multiply a two-dimensional extended precision matrices with a two dimensional extended precision vector. A new vector is returned which is the product of the matrix and the vector. The product is calculated using the well known matrix-vector multiplication algorithm.

64.4.205 multiply(Tmatrix2_single,single):Tmatrix2_single

Synopsis: Multiply a two-dimensional single precision matrix by a scalar.

Declaration: `operator *(const m: Tmatrix2_single; const x: single) : Tmatrix2_single`

Visibility: default

Description: This operator allows you to multiply a matrix with a scalar. All elements in the matrix are multiplied by the scalar, the result is returned as a new matrix.

64.4.206 multiply(Tmatrix2_single,Tmatrix2_single):Tmatrix2_single

Synopsis: Give product of two two-dimensional single precision matrices.

Declaration: `operator *(const m1: Tmatrix2_single; const m2: Tmatrix2_single)
: Tmatrix2_single`

Visibility: default

Description: This operator allows you to multiply two two-dimensional single precision matrices. A new matrix is returned which is the product of both matrices. The product is calculated using the well known matrix multiplication algorithm.

64.4.207 multiply(Tmatrix2_single,Tvector2_single):Tvector2_single

Synopsis: Give product of a two-dimensional single precision matrix and vector.

Declaration: `operator *(const m: Tmatrix2_single; const v: Tvector2_single)
: Tvector2_single`

Visibility: default

Description: This operator allows you to multiply a two-dimensional single precision matrices with a two dimensional single precision vector. A new vector is returned which is the product of the matrix and the vector. The product is calculated using the well known matrix-vector multiplication algorithm.

64.4.208 multiply(Tmatrix3_double,Double):Tmatrix3_double

Synopsis: Multiply a three-dimensional double precision matrix by a scalar.

Declaration: `operator *(const m: Tmatrix3_double; const x: Double) : Tmatrix3_double`

Visibility: default

Description: This operator allows you to multiply a matrix with a scalar. All elements in the matrix are multiplied by the scalar, the result is returned as a new matrix.

64.4.209 multiply(Tmatrix3_double,Tmatrix3_double):Tmatrix3_double

Synopsis: Give product of two three-dimensional double precision matrices.

Declaration: `operator *(const m1: Tmatrix3_double; const m2: Tmatrix3_double)
: Tmatrix3_double`

Visibility: default

Description: This operator allows you to multiply two two-dimensional single precision matrices. A new matrix is returned which is the product of both matrices. The product is calculated using the well known matrix multiplication algorithm.

64.4.210 multiply(Tmatrix3_double,Tvector3_double):Tvector3_double

Synopsis: Give product of a three-dimensional double precision matrix and vector.

Declaration: `operator *(const m: Tmatrix3_double; const v: Tvector3_double)
: Tvector3_double`

Visibility: default

Description: This operator allows you to multiply a three-dimensional double precision matrices with a three dimensional double precision vector. A new vector is returned which is the product of the matrix and the vector. The product is calculated using the well known matrix-vector multiplication algorithm.

64.4.211 multiply(Tmatrix3_extended,extended):Tmatrix3_extended

Synopsis: Multiply a three-dimensional extended precision matrix by a scalar.

Declaration: `operator *(const m: Tmatrix3_extended; const x: extended)
: Tmatrix3_extended`

Visibility: default

Description: This operator allows you to multiply a matrix with a scalar. All elements in the matrix are multiplied by the scalar, the result is returned as a new matrix.

64.4.212 multiply(Tmatrix3_extended,Tmatrix3_extended):Tmatrix3_extended

Synopsis: Give product of two three-dimensional extended precision matrices.

Declaration: `operator *(const m1: Tmatrix3_extended; const m2: Tmatrix3_extended)
: Tmatrix3_extended`

Visibility: default

Description: This operator allows you to multiply two two-dimensional single precision matrices. A new matrix is returned which is the product of both matrices. The product is calculated using the well known matrix multiplication algorithm.

64.4.213 multiply(Tmatrix3_extended,Tvector3_extended):Tvector3_extended

Synopsis: Give product of a three-dimensional extended precision matrix and vector.

Declaration: `operator *(const m: Tmatrix3_extended; const v: Tvector3_extended)
: Tvector3_extended`

Visibility: default

Description: This operator allows you to multiply a three-dimensional extended precision matrices with a three dimensional extended precision vector. A new vector is returned which is the product of the matrix and the vector. The product is calculated using the well known matrix-vector multiplication algorithm.

64.4.214 multiply(Tmatrix3_single,single):Tmatrix3_single

Synopsis: Multiply a three-dimensional single precision matrix by a scalar.

Declaration: `operator *(const m: Tmatrix3_single; const x: single) : Tmatrix3_single`

Visibility: default

Description: This operator allows you to multiply a matrix with a scalar. All elements in the matrix are multiplied by the scalar, the result is returned as a new matrix.

64.4.215 multiply(Tmatrix3_single,Tmatrix3_single):Tmatrix3_single

Synopsis: Give product of two three-dimensional single precision matrices.

Declaration: `operator *(const m1: Tmatrix3_single; const m2: Tmatrix3_single)
: Tmatrix3_single`

Visibility: default

Description: This operator allows you to multiply two two-dimensional single precision matrices. A new matrix is returned which is the product of both matrices. The product is calculated using the well known matrix multiplication algorithm.

64.4.216 multiply(Tmatrix3_single,Tvector3_single):Tvector3_single

Synopsis: Give product of a three-dimensional single precision matrix and vector.

Declaration: `operator *(const m: Tmatrix3_single; const v: Tvector3_single)
: Tvector3_single`

Visibility: default

Description: This operator allows you to multiply a three-dimensional single precision matrices with a three dimensional single precision vector. A new vector is returned which is the product of the matrix and the vector. The product is calculated using the well known matrix-vector multiplication algorithm.

64.4.217 multiply(Tmatrix4_double,Double):Tmatrix4_double

Synopsis: Multiply a four-dimensional double precision matrix by a scalar.

Declaration: `operator *(const m: Tmatrix4_double; const x: Double) : Tmatrix4_double`

Visibility: default

Description: This operator allows you to multiply a matrix with a scalar. All elements in the matrix are multiplied by the scalar, the result is returned as a new matrix.

64.4.218 multiply(Tmatrix4_double,Tmatrix4_double):Tmatrix4_double

Synopsis: Give product of two four-dimensional double precision matrices.

Declaration: `operator *(const m1: Tmatrix4_double; const m2: Tmatrix4_double)
: Tmatrix4_double`

Visibility: default

Description: This operator allows you to multiply two two-dimensional single precision matrices. A new matrix is returned which is the product of both matrices. The product is calculated using the well known matrix multiplication algorithm.

64.4.219 multiply(Tmatrix4_double,Tvector4_double):Tvector4_double

Synopsis: Give product of a four-dimensional double precision matrix and vector.

Declaration: `operator *(const m: Tmatrix4_double; const v: Tvector4_double)
: Tvector4_double`

Visibility: default

Description: This operator allows you to multiply a four-dimensional double precision matrices with a four dimensional double precision vector. A new vector is returned which is the product of the matrix and the vector. The product is calculated using the well known matrix-vector multiplication algorithm.

64.4.220 multiply(Tmatrix4_extended,extended):Tmatrix4_extended

Synopsis: Multiply a four-dimensional extended precision matrix by a scalar.

Declaration: `operator *(const m: Tmatrix4_extended; const x: extended)
: Tmatrix4_extended`

Visibility: default

Description: This operator allows you to multiply a matrix with a scalar. All elements in the matrix are multiplied by the scalar, the result is returned as a new matrix.

64.4.221 multiply(Tmatrix4_extended,Tmatrix4_extended):Tmatrix4_extended

Synopsis: Give product of two four-dimensional extended precision matrices.

Declaration: `operator *(const m1: Tmatrix4_extended; const m2: Tmatrix4_extended)
: Tmatrix4_extended`

Visibility: default

Description: This operator allows you to multiply two two-dimensional single precision matrices. A new matrix is returned which is the product of both matrices. The product is calculated using the well known matrix multiplication algorithm.

64.4.222 multiply(Tmatrix4_extended,Tvector4_extended):Tvector4_extended

Synopsis: Give product of a four-dimensional extended precision matrix and vector.

Declaration: `operator *(const m: Tmatrix4_extended; const v: Tvector4_extended)
: Tvector4_extended`

Visibility: default

Description: This operator allows you to multiply a four-dimensional extended precision matrices with a four dimensional extended precision vector. A new vector is returned which is the product of the matrix and the vector. The product is calculated using the well known matrix-vector multiplication algorithm.

64.4.223 multiply(Tmatrix4_single,single):Tmatrix4_single

Synopsis: Multiply a four-dimensional single precision matrix by a scalar.

Declaration: `operator *(const m: Tmatrix4_single; const x: single) : Tmatrix4_single`

Visibility: default

Description: This operator allows you to multiply a matrix with a scalar. All elements in the matrix are multiplied by the scalar, the result is returned as a new matrix.

64.4.224 multiply(Tmatrix4_single,Tmatrix4_single):Tmatrix4_single

Synopsis: Give product of two four-dimensional single precision matrices.

Declaration: `operator *(const m1: Tmatrix4_single; const m2: Tmatrix4_single)
: Tmatrix4_single`

Visibility: default

Description: This operator allows you to multiply two two-dimensional single precision matrices. A new matrix is returned which is the product of both matrices. The product is calculated using the well known matrix multiplication algorithm.

64.4.225 multiply(Tmatrix4_single,Tvector4_single):Tvector4_single

Synopsis: Give product of a four-dimensional single precision matrix and vector.

Declaration: `operator *(const m: Tmatrix4_single; const v: Tvector4_single)
: Tvector4_single`

Visibility: default

Description: This operator allows you to multiply a four-dimensional single precision matrices with a four dimensional single precision vector. A new vector is returned which is the product of the matrix and the vector. The product is calculated using the well known matrix-vector multiplication algorithm.

64.4.226 multiply(Tvector2_double,Double):Tvector2_double

Synopsis: Multiply a two-dimensional double precision vector by a scalar.

Declaration: `operator *(const x: Tvector2_double; y: Double) : Tvector2_double`

Visibility: default

Description: This operator allows you to multiply a vector by a scalar value. Each vector element is multiplied by the scalar value; the result is returned as a new vector.

64.4.227 multiply(Tvector2_double,Tvector2_double):Tvector2_double

Synopsis: Multiply two vectors element wise.

Declaration: `operator *(const x: Tvector2_double; const y: Tvector2_double)
: Tvector2_double`

Visibility: default

Description: This operator returns a vector that contains the element by element multiplication of the two multiplied vectors.

64.4.228 multiply(Tvector2_extended,extended):Tvector2_extended

Synopsis: Multiply a two-dimensional extended precision vector by a scalar.

Declaration: `operator *(const x: Tvector2_extended; y: extended) : Tvector2_extended`

Visibility: default

Description: This operator allows you to multiply a vector by a scalar value. Each vector element is multiplied by the scalar value; the result is returned as a new vector.

64.4.229 multiply(Tvector2_extended,Tvector2_extended):Tvector2_extended

Synopsis: Multiply two vectors element wise.

Declaration: `operator *(const x: Tvector2_extended; const y: Tvector2_extended) : Tvector2_extended`

Visibility: default

Description: This operator returns a vector that contains the element by element multiplication of the two multiplied vectors.

64.4.230 multiply(Tvector2_single,single):Tvector2_single

Synopsis: Multiply a two-dimensional single precision vector by a scalar.

Declaration: `operator *(const x: Tvector2_single; y: single) : Tvector2_single`

Visibility: default

Description: This operator allows you to multiply a vector by a scalar value. Each vector element is multiplied by the scalar value; the result is returned as a new vector.

64.4.231 multiply(Tvector2_single,Tvector2_single):Tvector2_single

Synopsis: Multiply two vectors element wise.

Declaration: `operator *(const x: Tvector2_single; const y: Tvector2_single) : Tvector2_single`

Visibility: default

Description: This operator returns a vector that contains the element by element multiplication of the two multiplied vectors.

64.4.232 multiply(Tvector3_double,Double):Tvector3_double

Synopsis: Multiply a three-dimensional double precision vector by a scalar.

Declaration: `operator *(const x: Tvector3_double; y: Double) : Tvector3_double`

Visibility: default

Description: This operator allows you to multiply a vector by a scalar value. Each vector element is multiplied by the scalar value; the result is returned as a new vector.

64.4.233 multiply(Tvector3_double,Tvector3_double):Tvector3_double

Synopsis: Multiply two vectors element wise.

Declaration: `operator *(const x: Tvector3_double; const y: Tvector3_double)
: Tvector3_double`

Visibility: default

Description: This operator returns a vector that contains the element by element multiplication of the two multiplied vectors.

64.4.234 multiply(Tvector3_extended,extended):Tvector3_extended

Synopsis: Multiply a three-dimensional extended precision vector by a scalar.

Declaration: `operator *(const x: Tvector3_extended; y: extended) : Tvector3_extended`

Visibility: default

Description: This operator allows you to multiply a vector by a scalar value. Each vector element is multiplied by the scalar value; the result is returned as a new vector.

64.4.235 multiply(Tvector3_extended,Tvector3_extended):Tvector3_extended

Synopsis: Multiply two vectors element wise.

Declaration: `operator *(const x: Tvector3_extended; const y: Tvector3_extended)
: Tvector3_extended`

Visibility: default

Description: This operator returns a vector that contains the element by element multiplication of the two multiplied vectors.

64.4.236 multiply(Tvector3_single,single):Tvector3_single

Synopsis: Multiply a three-dimensional single precision vector by a scalar.

Declaration: `operator *(const x: Tvector3_single; y: single) : Tvector3_single`

Visibility: default

Description: This operator allows you to multiply a vector by a scalar value. Each vector element is multiplied by the scalar value; the result is returned as a new vector.

64.4.237 multiply(Tvector3_single,Tvector3_single):Tvector3_single

Synopsis: Multiply two vectors element wise.

Declaration: `operator *(const x: Tvector3_single; const y: Tvector3_single)
: Tvector3_single`

Visibility: default

Description: This operator returns a vector that contains the element by element multiplication of the two multiplied vectors.

64.4.238 multiply(Tvector4_double,Double):Tvector4_double

Synopsis: Multiply a four-dimensional double precision vector by a scalar.

Declaration: `operator *(const x: Tvector4_double; y: Double) : Tvector4_double`

Visibility: default

Description: This operator allows you to multiply a vector by a scalar value. Each vector element is multiplied by the scalar value; the result is returned as a new vector.

64.4.239 multiply(Tvector4_double,Tvector4_double):Tvector4_double

Synopsis: Multiply two vectors element wise.

Declaration: `operator *(const x: Tvector4_double; const y: Tvector4_double)
: Tvector4_double`

Visibility: default

Description: This operator returns a vector that contains the element by element multiplication of the two multiplied vectors.

64.4.240 multiply(Tvector4_extended,extended):Tvector4_extended

Synopsis: Multiply a four-dimensional extended precision vector by a scalar.

Declaration: `operator *(const x: Tvector4_extended; y: extended) : Tvector4_extended`

Visibility: default

Description: This operator allows you to multiply a vector by a scalar value. Each vector element is multiplied by the scalar value; the result is returned as a new vector.

64.4.241 multiply(Tvector4_extended,Tvector4_extended):Tvector4_extended

Synopsis: Multiply two vectors element wise.

Declaration: `operator *(const x: Tvector4_extended; const y: Tvector4_extended)
: Tvector4_extended`

Visibility: default

Description: This operator returns a vector that contains the element by element multiplication of the two multiplied vectors.

64.4.242 multiply(Tvector4_single,single):Tvector4_single

Synopsis: Multiply a four-dimensional single precision vector by a scalar.

Declaration: `operator *(const x: Tvector4_single; y: single) : Tvector4_single`

Visibility: default

Description: This operator allows you to multiply a vector by a scalar value. Each vector element is multiplied by the scalar value; the result is returned as a new vector.

64.4.243 multiply(Tvector4_single,Tvector4_single):Tvector4_single

Synopsis: Multiply two vectors element wise.

Declaration: `operator *(const x: Tvector4_single; const y: Tvector4_single)
: Tvector4_single`

Visibility: default

Description: This operator returns a vector that contains the element by element multiplication of the two multiplied vectors.

64.4.244 negative(Tmatrix2_double):Tmatrix2_double

Synopsis: Negate two-dimensional double precision matrix.

Declaration: `operator -(const m1: Tmatrix2_double) : Tmatrix2_double`

Visibility: default

Description: This operation returns a matrix with all elements negated.

64.4.245 negative(Tmatrix2_extended):Tmatrix2_extended

Synopsis: Negate two-dimensional extended precision matrix.

Declaration: `operator -(const m1: Tmatrix2_extended) : Tmatrix2_extended`

Visibility: default

Description: This operation returns a matrix with all elements negated.

64.4.246 negative(Tmatrix2_single):Tmatrix2_single

Synopsis: Negate two-dimensional single precision matrix.

Declaration: `operator -(const m1: Tmatrix2_single) : Tmatrix2_single`

Visibility: default

Description: This operation returns a matrix with all elements negated.

64.4.247 negative(Tmatrix3_double):Tmatrix3_double

Synopsis: Negate three-dimensional double precision matrix.

Declaration: `operator -(const m1: Tmatrix3_double) : Tmatrix3_double`

Visibility: default

Description: This operation returns a matrix with all elements negated.

64.4.248 negative(Tmatrix3_extended):Tmatrix3_extended

Synopsis: Negate three-dimensional extended precision matrix.

Declaration: `operator -(const m1: Tmatrix3_extended) : Tmatrix3_extended`

Visibility: default

Description: This operation returns a matrix with all elements negated.

64.4.249 negative(Tmatrix3_single):Tmatrix3_single

Synopsis: Negate three-dimensional single precision matrix.

Declaration: `operator -(const m1: Tmatrix3_single) : Tmatrix3_single`

Visibility: default

Description: This operation returns a matrix with all elements negated.

64.4.250 negative(Tmatrix4_double):Tmatrix4_double

Synopsis: Negate four-dimensional double precision matrix.

Declaration: `operator -(const m1: Tmatrix4_double) : Tmatrix4_double`

Visibility: default

Description: This operation returns a matrix with all elements negated.

64.4.251 negative(Tmatrix4_extended):Tmatrix4_extended

Synopsis: Negate four-dimensional extended precision matrix.

Declaration: `operator -(const m1: Tmatrix4_extended) : Tmatrix4_extended`

Visibility: default

Description: This operation returns a matrix with all elements negated.

64.4.252 negative(Tmatrix4_single):Tmatrix4_single

Synopsis: Negate four-dimensional single precision matrix.

Declaration: `operator -(const m1: Tmatrix4_single) : Tmatrix4_single`

Visibility: default

Description: This operation returns a matrix with all elements negated.

64.4.253 negative(Tvector2_double):Tvector2_double

Synopsis: Negate two-dimensional vector.

Declaration: `operator -(const x: Tvector2_double) : Tvector2_double`

Visibility: default

Description: This operation returns a vector in the opposite direction of the vector that is passed. In order to do so, all values in the vector are negated.

64.4.254 negative(Tvector2_extended):Tvector2_extended

Synopsis: Negate two-dimensional vector.

Declaration: `operator -(const x: Tvector2_extended) : Tvector2_extended`

Visibility: default

Description: This operation returns a vector in the opposite direction of the vector that is passed. In order to do so, all values in the vector are negated.

64.4.255 negative(Tvector2_single):Tvector2_single

Synopsis: Negate two-dimensional vector.

Declaration: `operator -(const x: Tvector2_single) : Tvector2_single`

Visibility: default

Description: This operation returns a vector in the opposite direction of the vector that is passed. In order to do so, all values in the vector are negated.

64.4.256 negative(Tvector3_double):Tvector3_double

Synopsis: Negate three-dimensional vector.

Declaration: `operator -(const x: Tvector3_double) : Tvector3_double`

Visibility: default

Description: This operation returns a vector in the opposite direction of the vector that is passed. In order to do so, all values in the vector are negated.

64.4.257 negative(Tvector3_extended):Tvector3_extended

Synopsis: Negate three-dimensional vector.

Declaration: `operator -(const x: Tvector3_extended) : Tvector3_extended`

Visibility: default

Description: This operation returns a vector in the opposite direction of the vector that is passed. In order to do so, all values in the vector are negated.

64.4.258 negative(Tvector3_single):Tvector3_single

Synopsis: Negate three-dimensional vector.

Declaration: `operator -(const x: Tvector3_single) : Tvector3_single`

Visibility: default

Description: This operation returns a vector in the opposite direction of the vector that is passed. In order to do so, all values in the vector are negated.

64.4.259 negative(Tvector4_double):Tvector4_double

Synopsis: Negate four-dimensional vector.

Declaration: `operator -(const x: Tvector4_double) : Tvector4_double`

Visibility: default

Description: This operation returns a vector in the opposite direction of the vector that is passed. In order to do so, all values in the vector are negated.

64.4.260 negative(Tvector4_extended):Tvector4_extended

Synopsis: Negate four-dimensional vector.

Declaration: `operator -(const x: Tvector4_extended) : Tvector4_extended`

Visibility: default

Description: This operation returns a vector in the opposite direction of the vector that is passed. In order to do so, all values in the vector are negated.

64.4.261 negative(Tvector4_single):Tvector4_single

Synopsis: Negate four-dimensional vector.

Declaration: `operator -(const x: Tvector4_single) : Tvector4_single`

Visibility: default

Description: This operation returns a vector in the opposite direction of the vector that is passed. In order to do so, all values in the vector are negated.

64.4.262 power(Tvector2_double,Tvector2_double):Double

Synopsis: Calculate the internal product of two vectors.

Declaration: `operator ** (const x: Tvector2_double; const y: Tvector2_double) : Double`

Visibility: default

Description: This operator returns the internal product of the two vectors, that is, the elements of the two vectors are element-wise multiplied, and then added together.

64.4.263 power(Tvector2_extended,Tvector2_extended):extended

Synopsis: Calculate the internal product of two vectors.

Declaration: `operator ** (const x: Tvector2_extended; const y: Tvector2_extended)
: extended`

Visibility: default

Description: This operator returns the internal product of the two vectors, that is, the elements of the two vectors are element-wise multiplied, and then added together.

64.4.264 power(Tvector2_single,Tvector2_single):single

Synopsis: Calculate the internal product of two vectors.

Declaration: `operator ** (const x: Tvector2_single; const y: Tvector2_single) : single`

Visibility: default

Description: This operator returns the internal product of the two vectors, that is, the elements of the two vectors are element-wise multiplied, and then added together.

64.4.265 power(Tvector3_double,Tvector3_double):Double

Synopsis: Calculate the internal product of two vectors.

Declaration: `operator ** (const x: Tvector3_double; const y: Tvector3_double) : Double`

Visibility: default

Description: This operator returns the internal product of the two vectors, that is, the elements of the two vectors are element-wise multiplied, and then added together.

64.4.266 power(Tvector3_extended,Tvector3_extended):extended

Synopsis: Calculate the internal product of two vectors.

Declaration: `operator ** (const x: Tvector3_extended; const y: Tvector3_extended)
: extended`

Visibility: default

Description: This operator returns the internal product of the two vectors, that is, the elements of the two vectors are element-wise multiplied, and then added together.

64.4.267 power(Tvector3_single,Tvector3_single):single

Synopsis: Calculate the internal product of two vectors.

Declaration: `operator ** (const x: Tvector3_single; const y: Tvector3_single) : single`

Visibility: default

Description: This operator returns the internal product of the two vectors, that is, the elements of the two vectors are element-wise multiplied, and then added together.

64.4.268 power(Tvector4_double,Tvector4_double):Double

Synopsis: Calculate the internal product of two vectors.

Declaration: `operator ** (const x: Tvector4_double; const y: Tvector4_double) : Double`

Visibility: default

Description: This operator returns the internal product of the two vectors, that is, the elements of the two vectors are element-wise multiplied, and then added together.

64.4.269 power(Tvector4_extended,Tvector4_extended):extended

Synopsis: Calculate the internal product of two vectors.

Declaration: `operator ** (const x: Tvector4_extended; const y: Tvector4_extended)
: extended`

Visibility: default

Description: This operator returns the internal product of the two vectors, that is, the elements of the two vectors are element-wise multiplied, and then added together.

64.4.270 power(Tvector4_single,Tvector4_single):single

Synopsis: Calculate the internal product of two vectors.

Declaration: `operator ** (const x: Tvector4_single; const y: Tvector4_single) : single`

Visibility: default

Description: This operator returns the internal product of the two vectors, that is, the elements of the two vectors are element-wise multiplied, and then added together.

64.4.271 subtract(Tmatrix2_double,Double):Tmatrix2_double

Synopsis: Subtract scalar to two-dimensional double precision matrix.

Declaration: `operator - (const m: Tmatrix2_double; const x: Double) : Tmatrix2_double`

Visibility: default

Description: This operator allows you to subtract a scalar value from a matrix. The scalar is subtracted from all elements of the matrix, the result is returned as a new matrix.

64.4.272 subtract(Tmatrix2_double,Tmatrix2_double):Tmatrix2_double

Synopsis: Subtract a two-dimensional double precision matrix from another.

Declaration: `operator - (const m1: Tmatrix2_double; const m2: Tmatrix2_double)
: Tmatrix2_double`

Visibility: default

Description: This operator allows you to subtract a two-dimensional double precision matrix from another. A new matrix is returned with all elements of the two matrices subtracted from each other.

64.4.273 subtract(Tmatrix2_extended,extended):Tmatrix2_extended

Synopsis: Add scalar to two-dimensional extended precision matrix.

Declaration: `operator - (const m: Tmatrix2_extended; const x: extended)
: Tmatrix2_extended`

Visibility: default

Description: This operator allows you to subtract a scalar value from a matrix. The scalar is subtracted from all elements of the matrix, the result is returned as a new matrix.

64.4.274 subtract(Tmatrix2_extended,Tmatrix2_extended):Tmatrix2_extended

Synopsis: Subtract a two-dimensional extended precision matrix from another.

Declaration: `operator - (const m1: Tmatrix2_extended; const m2: Tmatrix2_extended)
: Tmatrix2_extended`

Visibility: default

Description: This operator allows you to subtract a two-dimensional extended precision matrix from another. A new matrix is returned with all elements of the two matrices subtracted from each other.

64.4.275 subtract(Tmatrix2_single,single):Tmatrix2_single

Synopsis: Subtract scalar to two-dimensional single precision matrix.

Declaration: `operator -(const m: Tmatrix2_single; const x: single) : Tmatrix2_single`

Visibility: default

Description: This operator allows you to subtract a scalar value from a matrix. The scalar is subtracted from all elements of the matrix, the result is returned as a new matrix.

64.4.276 subtract(Tmatrix2_single,Tmatrix2_single):Tmatrix2_single

Synopsis: Subtract a two-dimensional single precision matrix from another.

Declaration: `operator -(const m1: Tmatrix2_single; const m2: Tmatrix2_single)
: Tmatrix2_single`

Visibility: default

Description: This operator allows you to subtract a two-dimensional single precision matrix from another. A new matrix is returned with all elements of the two matrices subtracted from each other.

64.4.277 subtract(Tmatrix3_double,Double):Tmatrix3_double

Synopsis: Add scalar to three-dimensional double precision matrix.

Declaration: `operator -(const m: Tmatrix3_double; const x: Double) : Tmatrix3_double`

Visibility: default

Description: This operator allows you to subtract a scalar value from a matrix. The scalar is subtracted from all elements of the matrix, the result is returned as a new matrix.

64.4.278 subtract(Tmatrix3_double,Tmatrix3_double):Tmatrix3_double

Synopsis: Subtract a three-dimensional double precision matrix from another.

Declaration: `operator -(const m1: Tmatrix3_double; const m2: Tmatrix3_double)
: Tmatrix3_double`

Visibility: default

Description: This operator allows you to subtract a three-dimensional double precision matrix from another. A new matrix is returned with all elements of the two matrices subtracted from each other.

64.4.279 subtract(Tmatrix3_extended,extended):Tmatrix3_extended

Synopsis: Add scalar to three-dimensional extended precision matrix.

Declaration: `operator -(const m: Tmatrix3_extended; const x: extended)
: Tmatrix3_extended`

Visibility: default

Description: This operator allows you to subtract a scalar value from a matrix. The scalar is subtracted from all elements of the matrix, the result is returned as a new matrix.

64.4.280 subtract(Tmatrix3_extended,Tmatrix3_extended):Tmatrix3_extended

Synopsis: Subtract a three-dimensional extended precision matrix from another.

Declaration: `operator -(const m1: Tmatrix3_extended; const m2: Tmatrix3_extended)
: Tmatrix3_extended`

Visibility: default

Description: This operator allows you to subtract a three-dimensional extended precision matrix from another. A new matrix is returned with all elements of the two matrices subtracted from each other.

64.4.281 subtract(Tmatrix3_single,single):Tmatrix3_single

Synopsis: Add scalar to three-dimensional single precision matrix.

Declaration: `operator -(const m: Tmatrix3_single; const x: single) : Tmatrix3_single`

Visibility: default

Description: This operator allows you to subtract a scalar value from a matrix. The scalar is subtracted from all elements of the matrix, the result is returned as a new matrix.

64.4.282 subtract(Tmatrix3_single,Tmatrix3_single):Tmatrix3_single

Synopsis: Subtract a three-dimensional single precision matrix from another.

Declaration: `operator -(const m1: Tmatrix3_single; const m2: Tmatrix3_single)
: Tmatrix3_single`

Visibility: default

Description: This operator allows you to subtract a three-dimensional single precision matrix from another. A new matrix is returned with all elements of the two matrices subtracted from each other.

64.4.283 subtract(Tmatrix4_double,Double):Tmatrix4_double

Synopsis: Add scalar to four-dimensional double precision matrix.

Declaration: `operator -(const m: Tmatrix4_double; const x: Double) : Tmatrix4_double`

Visibility: default

Description: This operator allows you to subtract a scalar value from a matrix. The scalar is subtracted from all elements of the matrix, the result is returned as a new matrix.

64.4.284 subtract(Tmatrix4_double,Tmatrix4_double):Tmatrix4_double

Synopsis: Subtract a four-dimensional double precision matrix from another.

Declaration: `operator -(const m1: Tmatrix4_double; const m2: Tmatrix4_double)
: Tmatrix4_double`

Visibility: default

Description: This operator allows you to subtract a four-dimensional double precision matrix from another. A new matrix is returned with all elements of the two matrices subtracted from each other.

64.4.285 subtract(Tmatrix4_extended,extended):Tmatrix4_extended

Synopsis: Add scalar to four-dimensional extended precision matrix.

Declaration: `operator -(const m: Tmatrix4_extended; const x: extended)
: Tmatrix4_extended`

Visibility: default

Description: This operator allows you to subtract a scalar value from a matrix. The scalar is subtracted from all elements of the matrix, the result is returned as a new matrix.

64.4.286 subtract(Tmatrix4_extended,Tmatrix4_extended):Tmatrix4_extended

Synopsis: Subtract a four-dimensional extended precision matrix from another.

Declaration: `operator -(const m1: Tmatrix4_extended; const m2: Tmatrix4_extended)
: Tmatrix4_extended`

Visibility: default

Description: This operator allows you to subtract a four-dimensional extended precision matrix from another. A new matrix is returned with all elements of the two matrices subtracted from each other.

64.4.287 subtract(Tmatrix4_single,single):Tmatrix4_single

Synopsis: Add scalar to four-dimensional single precision matrix.

Declaration: `operator -(const m: Tmatrix4_single; const x: single) : Tmatrix4_single`

Visibility: default

Description: This operator allows you to subtract a scalar value from a matrix. The scalar is subtracted from all elements of the matrix, the result is returned as a new matrix.

64.4.288 subtract(Tmatrix4_single,Tmatrix4_single):Tmatrix4_single

Synopsis: Subtract a four-dimensional single precision matrix from another.

Declaration: `operator -(const m1: Tmatrix4_single; const m2: Tmatrix4_single)
: Tmatrix4_single`

Visibility: default

Description: This operator allows you to subtract a four-dimensional single precision matrix from another. A new matrix is returned with all elements of the two matrices subtracted from each other.

64.4.289 subtract(Tvector2_double,Double):Tvector2_double

Synopsis: Subtract scalar from two-dimensional double precision vector.

Declaration: `operator -(const x: Tvector2_double; y: Double) : Tvector2_double`

Visibility: default

Description: This operator allows you to subtract a scalar value from a vector. The scalar is subtracted from all elements of the vector, the result is returned as a new vector.

64.4.290 subtract(Tvector2_double,Tvector2_double):Tvector2_double

Synopsis: Subtract two-dimensional double precision vectors from each other.

Declaration: `operator -(const x: Tvector2_double; const y: Tvector2_double)
: Tvector2_double`

Visibility: default

Description: This operator allows you to subtract two two-dimensional vectors with double precision from each other. The result is a new vector which consists of the difference of the individual elements of the two vectors.

64.4.291 subtract(Tvector2_extended,extended):Tvector2_extended

Synopsis: Subtract scalar from two-dimensional extended precision vector.

Declaration: `operator -(const x: Tvector2_extended; y: extended) : Tvector2_extended`

Visibility: default

Description: This operator allows you to subtract a scalar value from a vector. The scalar is subtracted from all elements of the vector, the result is returned as a new vector.

64.4.292 subtract(Tvector2_extended,Tvector2_extended):Tvector2_extended

Synopsis: Subtract two-dimensional extended precision vectors from each other.

Declaration: `operator -(const x: Tvector2_extended; const y: Tvector2_extended)
: Tvector2_extended`

Visibility: default

Description: This operator allows you to subtract two two-dimensional vectors with extended precision from each other. The result is a new vector which consists of the difference of the individual elements of the two vectors.

64.4.293 subtract(Tvector2_single,single):Tvector2_single

Synopsis: Subtract scalar from two-dimensional single precision vector.

Declaration: `operator -(const x: Tvector2_single; y: single) : Tvector2_single`

Visibility: default

Description: This operator allows you to subtract a scalar value from a vector. The scalar is subtracted from all elements of the vector, the result is returned as a new vector.

64.4.294 subtract(Tvector2_single,Tvector2_single):Tvector2_single

Synopsis: Subtract two-dimensional single precision vectors from each other.

Declaration: `operator -(const x: Tvector2_single; const y: Tvector2_single)
: Tvector2_single`

Visibility: default

Description: This operator allows you to subtract two two-dimensional vectors with single precision from each other. The result is a new vector which consists of the difference of the individual elements of the two vectors.

64.4.295 subtract(Tvector3_double,Double):Tvector3_double

Synopsis: Subtract scalar from three-dimensional double precision vector.

Declaration: `operator -(const x: Tvector3_double; y: Double) : Tvector3_double`

Visibility: default

Description: This operator allows you to subtract a scalar value from a vector. The scalar is subtracted from all elements of the vector, the result is returned as a new vector.

64.4.296 subtract(Tvector3_double,Tvector3_double):Tvector3_double

Synopsis: Subtract three-dimensional double precision vectors from each other.

Declaration: `operator -(const x: Tvector3_double; const y: Tvector3_double)
: Tvector3_double`

Visibility: default

Description: This operator allows you to subtract two two-dimensional vectors with double precision from each other. The result is a new vector which consists of the difference of the individual elements of the two vectors.

64.4.297 subtract(Tvector3_extended,extended):Tvector3_extended

Synopsis: Subtract scalar from three-dimensional extended precision vector.

Declaration: `operator -(const x: Tvector3_extended; y: extended) : Tvector3_extended`

Visibility: default

Description: This operator allows you to subtract a scalar value from a vector. The scalar is subtracted from all elements of the vector, the result is returned as a new vector.

64.4.298 subtract(Tvector3_extended,Tvector3_extended):Tvector3_extended

Synopsis: Subtract three-dimensional extended precision vectors from each other.

Declaration: `operator -(const x: Tvector3_extended; const y: Tvector3_extended)
: Tvector3_extended`

Visibility: default

Description: This operator allows you to subtract two three-dimensional vectors with extended precision from each other. The result is a new vector which consists of the difference of the individual elements of the two vectors.

64.4.299 subtract(Tvector3_single,single):Tvector3_single

Synopsis: Subtract scalar from three-dimensional single precision vector.

Declaration: `operator -(const x: Tvector3_single; y: single) : Tvector3_single`

Visibility: default

Description: This operator allows you to subtract a scalar value from a vector. The scalar is subtracted from all elements of the vector, the result is returned as a new vector.

64.4.300 subtract(Tvector3_single,Tvector3_single):Tvector3_single

Synopsis: Subtract three-dimensional single precision vectors from each other.

Declaration: `operator -(const x: Tvector3_single; const y: Tvector3_single)
: Tvector3_single`

Visibility: default

Description: This operator allows you to subtract two three-dimensional vectors with single precision from each other. The result is a new vector which consists of the difference of the individual elements of the two vectors.

64.4.301 subtract(Tvector4_double,Double):Tvector4_double

Synopsis: Subtract scalar from four-dimensional double precision vector.

Declaration: `operator -(const x: Tvector4_double; y: Double) : Tvector4_double`

Visibility: default

Description: This operator allows you to subtract a scalar value from a vector. The scalar is subtracted from all elements of the vector, the result is returned as a new vector.

64.4.302 subtract(Tvector4_double,Tvector4_double):Tvector4_double

Synopsis: Subtract four-dimensional double precision vectors from each other.

Declaration: `operator -(const x: Tvector4_double; const y: Tvector4_double)
: Tvector4_double`

Visibility: default

Description: This operator allows you to subtract two four-dimensional vectors with double precision from each other. The result is a new vector which consists of the difference of the individual elements of the two vectors.

64.4.303 subtract(Tvector4_extended,extended):Tvector4_extended

Synopsis: Subtract scalar from four-dimensional extended precision vector.

Declaration: `operator -(const x: Tvector4_extended; y: extended) : Tvector4_extended`

Visibility: default

Description: This operator allows you to subtract a scalar value from a vector. The scalar is subtracted from all elements of the vector, the result is returned as a new vector.

64.4.304 subtract(Tvector4_extended,Tvector4_extended):Tvector4_extended

Synopsis: Subtract four-dimensional extended precision vectors from each other.

Declaration: `operator -(const x: Tvector4_extended; const y: Tvector4_extended)
: Tvector4_extended`

Visibility: default

Description: This operator allows you to subtract two four-dimensional vectors with extended precision from each other. The result is a new vector which consists of the difference of the individual elements of the two vectors.

64.4.305 subtract(Tvector4_single,single):Tvector4_single

Synopsis: Subtract scalar from four-dimensional single precision vector.

Declaration: `operator -(const x: Tvector4_single; y: single) : Tvector4_single`

Visibility: default

Description: This operator allows you to subtract a scalar value from a vector. The scalar is subtracted from all elements of the vector, the result is returned as a new vector.

64.4.306 subtract(Tvector4_single,Tvector4_single):Tvector4_single

Synopsis: Subtract four-dimensional single precision vectors from each other.

Declaration: `operator -(const x: Tvector4_single; const y: Tvector4_single)
: Tvector4_single`

Visibility: default

Description: This operator allows you to subtract two four-dimensional vectors with single precision from each other. The result is a new vector which consists of the difference of the individual elements of the two vectors.

64.4.307 symmetricaldifference(Tvector3_double,Tvector3_double):Tvector3_double

Synopsis: Calculate the external product of two three-dimensional vectors.

Declaration: `operator ><(const x: Tvector3_double; const y: Tvector3_double)
: Tvector3_double`

Visibility: default

Description: This operator returns the external product of two three dimensional vector. It is a vector orthonormal to the two multiplied vectors. The length of that vector is equal to the surface area of a parallelogram with the two vectors as sides.

The external product is often used to get a vector orthonormal to two other vectors, but of a predefined length. In order to do so, the result vector from the external product, is divided by its length, and then multiplied by the desired size.

64.4.308 symmetricaldifference(Tvector3_extended,Tvector3_extended):Tvector3_extended

Synopsis: Calculate the external product of two three-dimensional vectors.

Declaration: `operator ><(const x: Tvector3_extended; const y: Tvector3_extended)
: Tvector3_extended`

Visibility: default

Description: This operator returns the external product of two three dimensional vector. It is a vector orthonormal to the two multiplied vectors. The length of that vector is equal to the surface area of a parallelogram with the two vectors as sides.

The external product is often used to get a vector orthonormal to two other vectors, but of a predefined length. In order to do so, the result vector from the external product, is divided by its length, and then multiplied by the desired size.

64.4.309 symmetricaldifference(Tvector3_single,Tvector3_single):Tvector3_single

Synopsis: Calculate the external product of two three-dimensional vectors.

Declaration: `operator ><(const x: Tvector3_single; const y: Tvector3_single)
: Tvector3_single`

Visibility: default

Description: This operator returns the external product of two three dimensional vector. It is a vector orthonormal to the two multiplied vectors. The length of that vector is equal to the surface area of a parallelogram with the two vectors as sides.

The external product is often used to get a vector orthonormal to two other vectors, but of a predefined length. In order to do so, the result vector from the external product, is divided by its length, and then multiplied by the desired size.

64.5 Tmatrix2_double**64.5.1 Description**

The `Tmatrix2_double` object provides a matrix of 2*2 double precision scalars.

64.5.2 Method overview

Page	Method	Description
1153	<code>determinant</code>	Calculates the determinant of the matrix.
1153	<code>get_column</code>	Returns the c-th column of the matrix as vector.
1153	<code>get_row</code>	Returns the r-th row of the matrix as vector.
1153	<code>init</code>	Initializes the matrix, setting its elements to the values passed to the constructor.
1152	<code>init_identity</code>	Initializes the matrix and sets its elements to the identity matrix.
1152	<code>init_zero</code>	Initializes the matrix and sets its elements to zero.
1154	<code>inverse</code>	Calculates the inverse of the matrix.
1153	<code>set_column</code>	Sets c-th column of the matrix with a vector.
1153	<code>set_row</code>	Sets r-th row of the matrix with a vector.
1154	<code>transpose</code>	Returns the transposition of the matrix.

64.5.3 Tmatrix2_double.init_zero

Synopsis: Initializes the matrix and sets its elements to zero.

Declaration: `constructor init_zero`

Visibility: default

64.5.4 Tmatrix2_double.init_identity

Synopsis: Initializes the matrix and sets its elements to the identity matrix.

Declaration: `constructor init_identity`

Visibility: default

Description: Initializes the matrix and sets its elements to the identity matrix, that is, elements to 1 on the left-upper to right-lower diagonal, the rest zero.

64.5.5 Tmatrix2_double.init

Synopsis: Initializes the matrix, setting its elements to the values passed to the constructor.

Declaration: `constructor init(aa: Double; ab: Double; ba: Double; bb: Double)`

Visibility: default

Description: Initializes the matrix, setting its elements to the values passed to the constructor. The order of the values is left to right, then top to bottom.

64.5.6 Tmatrix2_double.get_column

Synopsis: Returns the *c*-th column of the matrix as vector.

Declaration: `function get_column(c: Byte) : Tvector2_double`

Visibility: default

Description: Returns the *c*-th column of the matrix as vector. The column numbering starts at 0.

64.5.7 Tmatrix2_double.get_row

Synopsis: Returns the *r*-th row of the matrix as vector.

Declaration: `function get_row(r: Byte) : Tvector2_double`

Visibility: default

Description: Returns the *r*-th row of the matrix as vector. The row numbering starts at 0.

64.5.8 Tmatrix2_double.set_column

Synopsis: Sets *c*-th column of the matrix with a vector.

Declaration: `procedure set_column(c: Byte; const v: Tvector2_double)`

Visibility: default

Description: Replaces the *c*-th column of the matrix with vector *v*. The column numbering starts at 0.

64.5.9 Tmatrix2_double.set_row

Synopsis: Sets *r*-th row of the matrix with a vector.

Declaration: `procedure set_row(r: Byte; const v: Tvector2_double)`

Visibility: default

Description: Replaces the *r*-th row of the matrix with vector *v*. The row numbering starts at 0.

64.5.10 Tmatrix2_double.determinant

Synopsis: Calculates the determinant of the matrix.

Declaration: `function determinant : Double`

Visibility: default

Description: Returns the determinant of the matrix.

64.5.11 Tmatrix2_double.inverse

Synopsis: Calculates the inverse of the matrix.

Declaration: `function inverse(Adeterminant: Double) : Tmatrix2_double`

Visibility: default

Description: `Tmatrix2_double.inverse` returns a new matrix that is the inverse of the matrix. You must pass the determinant of the matrix as parameter.

64.5.12 Tmatrix2_double.transpose

Synopsis: Returns the transposition of the matrix.

Declaration: `function transpose : Tmatrix2_double`

Visibility: default

Description: `Tmatrix2_double.transpose` returns a new matrix that is the transposition of the matrix, that is, the matrix with the x and y coordinates of the values swapped.

64.6 Tmatrix2_extended

64.6.1 Description

The `Tmatrix2_extended` object provides a matrix of 2*2 extended precision scalars.

64.6.2 Method overview

Page	Method	Description
1156	<code>determinant</code>	Calculates the determinant of the matrix.
1155	<code>get_column</code>	Returns the c-th column of the matrix as vector.
1155	<code>get_row</code>	Returns the r-th row of the matrix as vector.
1155	<code>init</code>	Initializes the matrix, setting its elements to the values passed to the constructor.
1155	<code>init_identity</code>	Initializes the matrix and sets its elements to the identity matrix.
1154	<code>init_zero</code>	Initializes the matrix and sets its elements to zero.
1156	<code>inverse</code>	Calculates the inverse of the matrix.
1155	<code>set_column</code>	Sets c-th column of the matrix with a vector.
1155	<code>set_row</code>	Sets r-th row of the matrix with a vector.
1156	<code>transpose</code>	Returns the transposition of the matrix.

64.6.3 Tmatrix2_extended.init_zero

Synopsis: Initializes the matrix and sets its elements to zero.

Declaration: `constructor init_zero`

Visibility: default

64.6.4 Tmatrix2_extended.init_identity

Synopsis: Initializes the matrix and sets its elements to the identity matrix.

Declaration: `constructor init_identity`

Visibility: default

Description: Initializes the matrix and sets its elements to the identity matrix, that is, elements to 1 on the left-upper to right-lower diagonal, the rest zero.

64.6.5 Tmatrix2_extended.init

Synopsis: Initializes the matrix, setting its elements to the values passed to the constructor.

Declaration: `constructor init(aa: extended; ab: extended; ba: extended; bb: extended)`

Visibility: default

Description: Initializes the matrix, setting its elements to the values passed to the constructor. The order of the values is left to right, then top to bottom.

64.6.6 Tmatrix2_extended.get_column

Synopsis: Returns the c-th column of the matrix as vector.

Declaration: `function get_column(c: Byte) : Tvector2_extended`

Visibility: default

Description: Returns the c-th column of the matrix as vector. The column numbering starts at 0.

64.6.7 Tmatrix2_extended.get_row

Synopsis: Returns the r-th row of the matrix as vector.

Declaration: `function get_row(r: Byte) : Tvector2_extended`

Visibility: default

Description: Returns the r-th row of the matrix as vector. The row numbering starts at 0.

64.6.8 Tmatrix2_extended.set_column

Synopsis: Sets c-th column of the matrix with a vector.

Declaration: `procedure set_column(c: Byte; const v: Tvector2_extended)`

Visibility: default

Description: Replaces the c-th column of the matrix with vector v. The column numbering starts at 0.

64.6.9 Tmatrix2_extended.set_row

Synopsis: Sets r-th row of the matrix with a vector.

Declaration: `procedure set_row(r: Byte; const v: Tvector2_extended)`

Visibility: default

Description: Replaces the r-th row of the matrix with vector v. The row numbering starts at 0.

64.6.10 Tmatrix2_extended.determinant

Synopsis: Calculates the determinant of the matrix.

Declaration: `function determinant : extended`

Visibility: default

Description: Returns the determinant of the matrix.

64.6.11 Tmatrix2_extended.inverse

Synopsis: Calculates the inverse of the matrix.

Declaration: `function inverse (Adeterminant: extended) : Tmatrix2_extended`

Visibility: default

Description: `Tmatrix2_extended.inverse` returns a new matrix that is the inverse of the matrix. You must pass the determinant of the matrix as parameter.

64.6.12 Tmatrix2_extended.transpose

Synopsis: Returns the transposition of the matrix.

Declaration: `function transpose : Tmatrix2_extended`

Visibility: default

Description: `Tmatrix2_extended.transpose` returns a new matrix that is the transposition of the matrix, that is, the matrix with the x and y coordinates of the values swapped.

64.7 Tmatrix2_single

64.7.1 Description

The `Tmatrix2_single` object provides a matrix of 2*2 single precision scalars.

64.7.2 Method overview

Page	Method	Description
1158	<code>determinant</code>	Calculates the determinant of the matrix.
1157	<code>get_column</code>	Returns the c-th column of the matrix as vector.
1157	<code>get_row</code>	Returns the r-th row of the matrix as vector.
1157	<code>init</code>	Initializes the matrix, setting its elements to the values passed to the constructor.
1157	<code>init_identity</code>	Initializes the matrix and sets its elements to the identity matrix.
1157	<code>init_zero</code>	Initializes the matrix and sets its elements to zero.
1158	<code>inverse</code>	Calculates the inverse of the matrix.
1157	<code>set_column</code>	Sets c-th column of the matrix with a vector.
1158	<code>set_row</code>	Sets r-th row of the matrix with a vector.
1158	<code>transpose</code>	Returns the transposition of the matrix.

64.7.3 Tmatrix2_single.init_zero

Synopsis: Initializes the matrix and sets its elements to zero.

Declaration: `constructor init_zero`

Visibility: default

64.7.4 Tmatrix2_single.init_identity

Synopsis: Initializes the matrix and sets its elements to the identity matrix.

Declaration: `constructor init_identity`

Visibility: default

Description: Initializes the matrix and sets its elements to the identity matrix, that is, elements to 1 on the left-upper to right-lower diagonal, the rest zero.

64.7.5 Tmatrix2_single.init

Synopsis: Initializes the matrix, setting its elements to the values passed to the constructor.

Declaration: `constructor init(aa: single; ab: single; ba: single; bb: single)`

Visibility: default

Description: Initializes the matrix, setting its elements to the values passed to the constructor. The order of the values is left to right, then top to bottom.

64.7.6 Tmatrix2_single.get_column

Synopsis: Returns the *c*-th column of the matrix as vector.

Declaration: `function get_column(c: Byte) : Tvector2_single`

Visibility: default

Description: Returns the *c*-th column of the matrix as vector. The column numbering starts at 0.

64.7.7 Tmatrix2_single.get_row

Synopsis: Returns the *r*-th row of the matrix as vector.

Declaration: `function get_row(r: Byte) : Tvector2_single`

Visibility: default

Description: Returns the *r*-th row of the matrix as vector. The row numbering starts at 0.

64.7.8 Tmatrix2_single.set_column

Synopsis: Sets *c*-th column of the matrix with a vector.

Declaration: `procedure set_column(c: Byte; const v: Tvector2_single)`

Visibility: default

Description: Replaces the *c*-th column of the matrix with vector *v*. The column numbering starts at 0.

64.7.9 Tmatrix2_single.set_row

Synopsis: Sets r-th row of the matrix with a vector.

Declaration: `procedure set_row(r: Byte; const v: Tvector2_single)`

Visibility: default

Description: Replaces the r-th row of the matrix with vector v. The row numbering starts at 0.

64.7.10 Tmatrix2_single.determinant

Synopsis: Calculates the determinant of the matrix.

Declaration: `function determinant : single`

Visibility: default

Description: Returns the determinant of the matrix.

64.7.11 Tmatrix2_single.inverse

Synopsis: Calculates the inverse of the matrix.

Declaration: `function inverse(A: Tmatrix2_single) : Tmatrix2_single`

Visibility: default

Description: `Tmatrix2_single.inverse` returns a new matrix that is the inverse of the matrix. You must pass the determinant of the matrix as parameter.

64.7.12 Tmatrix2_single.transpose

Synopsis: Returns the transposition of the matrix.

Declaration: `function transpose : Tmatrix2_single`

Visibility: default

Description: `Tmatrix2_single.transpose` returns a new matrix that is the transposition of the matrix, that is, the matrix with the x and y coordinates of the values swapped.

64.8 Tmatrix3_double**64.8.1 Description**

The `Tmatrix3_double` object provides a matrix of 3*3 double precision scalars.

64.8.2 Method overview

Page	Method	Description
1160	<code>determinant</code>	Calculates the determinant of the matrix.
1159	<code>get_column</code>	Returns the c-th column of the matrix as vector.
1160	<code>get_row</code>	Returns the r-th row of the matrix as vector.
1159	<code>init</code>	Initializes the matrix, setting its elements to the values passed to the constructor.
1159	<code>init_identity</code>	Initializes the matrix and sets its elements to the identity matrix.
1159	<code>init_zero</code>	Initializes the matrix and sets its elements to zero.
1160	<code>inverse</code>	Calculates the inverse of the matrix.
1160	<code>set_column</code>	Sets c-th column of the matrix with a vector.
1160	<code>set_row</code>	Sets r-th row of the matrix with a vector.
1160	<code>transpose</code>	Returns the transposition of the matrix.

64.8.3 Tmatrix3_double.init_zero

Synopsis: Initializes the matrix and sets its elements to zero.

Declaration: `constructor init_zero`

Visibility: default

64.8.4 Tmatrix3_double.init_identity

Synopsis: Initializes the matrix and sets its elements to the identity matrix.

Declaration: `constructor init_identity`

Visibility: default

Description: Initializes the matrix and sets its elements to the identity matrix, that is, elements to 1 on the left-upper to right-lower diagonal, the rest zero.

64.8.5 Tmatrix3_double.init

Synopsis: Initializes the matrix, setting its elements to the values passed to the constructor.

Declaration: `constructor init(aa: Double; ab: Double; ac: Double; ba: Double;
bb: Double; bc: Double; ca: Double; cb: Double;
cc: Double)`

Visibility: default

Description: Initializes the matrix, setting its elements to the values passed to the constructor. The order of the values is left to right, then top to bottom.

64.8.6 Tmatrix3_double.get_column

Synopsis: Returns the c-th column of the matrix as vector.

Declaration: `function get_column(c: Byte) : Tvector3_double`

Visibility: default

Description: Returns the c-th column of the matrix as vector. The column numbering starts at 0.

64.8.7 Tmatrix3_double.get_row

Synopsis: Returns the *r*-th row of the matrix as vector.

Declaration: `function get_row(r: Byte) : Tvector3_double`

Visibility: default

Description: Returns the *r*-th row of the matrix as vector. The row numbering starts at 0.

64.8.8 Tmatrix3_double.set_column

Synopsis: Sets *c*-th column of the matrix with a vector.

Declaration: `procedure set_column(c: Byte; const v: Tvector3_double)`

Visibility: default

Description: Replaces the *c*-th column of the matrix with vector *v*. The column numbering starts at 0.

64.8.9 Tmatrix3_double.set_row

Synopsis: Sets *r*-th row of the matrix with a vector.

Declaration: `procedure set_row(r: Byte; const v: Tvector3_double)`

Visibility: default

Description: Replaces the *r*-th row of the matrix with vector *v*. The row numbering starts at 0.

64.8.10 Tmatrix3_double.determinant

Synopsis: Calculates the determinant of the matrix.

Declaration: `function determinant : Double`

Visibility: default

Description: Returns the determinant of the matrix.

64.8.11 Tmatrix3_double.inverse

Synopsis: Calculates the inverse of the matrix.

Declaration: `function inverse(A_determinant: Double) : Tmatrix3_double`

Visibility: default

Description: `Tmatrix3_double.inverse` returns a new matrix that is the inverse of the matrix. You must pass the determinant of the matrix as parameter.

64.8.12 Tmatrix3_double.transpose

Synopsis: Returns the transposition of the matrix.

Declaration: `function transpose : Tmatrix3_double`

Visibility: default

Description: `Tmatrix2_double.transpose` returns a new matrix that is the transposition of the matrix, that is, the matrix with the *x* and *y* coordinates of the values swapped.

64.9 Tmatrix3_extended

64.9.1 Description

The `Tmatrix3_extended` object provides a matrix of 3*3 extended precision scalars.

64.9.2 Method overview

Page	Method	Description
1162	<code>determinant</code>	Calculates the determinant of the matrix.
1162	<code>get_column</code>	Returns the c-th column of the matrix as vector.
1162	<code>get_row</code>	Returns the r-th row of the matrix as vector.
1161	<code>init</code>	Initializes the matrix, setting its elements to the values passed to the constructor.
1161	<code>init_identity</code>	Initializes the matrix and sets its elements to the identity matrix.
1161	<code>init_zero</code>	Initializes the matrix and sets its elements to zero.
1162	<code>inverse</code>	Calculates the inverse of the matrix.
1162	<code>set_column</code>	Sets r-th column of the matrix with a vector.
1162	<code>set_row</code>	Sets r-th row of the matrix with a vector.
1163	<code>transpose</code>	Returns the transposition of the matrix.

64.9.3 Tmatrix3_extended.init_zero

Synopsis: Initializes the matrix and sets its elements to zero.

Declaration: `constructor init_zero`

Visibility: default

64.9.4 Tmatrix3_extended.init_identity

Synopsis: Initializes the matrix and sets its elements to the identity matrix.

Declaration: `constructor init_identity`

Visibility: default

Description: Initializes the matrix and sets its elements to the identity matrix, that is, elements to 1 on the left-upper to right-lower diagonal, the rest zero.

64.9.5 Tmatrix3_extended.init

Synopsis: Initializes the matrix, setting its elements to the values passed to the constructor.

Declaration: `constructor init(aa: extended; ab: extended; ac: extended;
ba: extended; bb: extended; bc: extended; ca: extended;
cb: extended; cc: extended)`

Visibility: default

Description: Initializes the matrix, setting its elements to the values passed to the constructor. The order of the values is left to right, then top to bottom.

64.9.6 Tmatrix3_extended.get_column

Synopsis: Returns the *c*-th column of the matrix as vector.

Declaration: `function get_column(c: Byte) : Tvector3_extended`

Visibility: default

Description: Returns the *c*-th column of the matrix as vector. The column numbering starts at 0.

64.9.7 Tmatrix3_extended.get_row

Synopsis: Returns the *r*-th row of the matrix as vector.

Declaration: `function get_row(r: Byte) : Tvector3_extended`

Visibility: default

Description: Returns the *r*-th row of the matrix as vector. The row numbering starts at 0.

64.9.8 Tmatrix3_extended.set_column

Synopsis: Sets *r*-th column of the matrix with a vector.

Declaration: `procedure set_column(c: Byte; const v: Tvector3_extended)`

Visibility: default

Description: Replaces the *c*-th column of the matrix with vector *v*. The column numbering starts at 0.

64.9.9 Tmatrix3_extended.set_row

Synopsis: Sets *r*-th row of the matrix with a vector.

Declaration: `procedure set_row(r: Byte; const v: Tvector3_extended)`

Visibility: default

Description: Replaces the *r*-th row of the matrix with vector *v*. The row numbering starts at 0.

64.9.10 Tmatrix3_extended.determinant

Synopsis: Calculates the determinant of the matrix.

Declaration: `function determinant : extended`

Visibility: default

Description: Returns the determinant of the matrix.

64.9.11 Tmatrix3_extended.inverse

Synopsis: Calculates the inverse of the matrix.

Declaration: `function inverse(Adeterminant: extended) : Tmatrix3_extended`

Visibility: default

Description: `Tmatrix3_extended.inverse` returns a new matrix that is the inverse of the matrix. You must pass the determinant of the matrix as parameter.

64.9.12 Tmatrix3_extended.transpose

Synopsis: Returns the transposition of the matrix.

Declaration: `function transpose : Tmatrix3_extended`

Visibility: default

Description: `Tmatrix2_extended.transpose` returns a new matrix that is the transposition of the matrix, that is, the matrix with the x and y coordinates of the values swapped.

64.10 Tmatrix3_single

64.10.1 Description

The `Tmatrix3_single` object provides a matrix of 3*3 single precision scalars.

64.10.2 Method overview

Page	Method	Description
1165	<code>determinant</code>	Calculates the determinant of the matrix.
1164	<code>get_column</code>	Returns the c-th column of the matrix as vector.
1164	<code>get_row</code>	Returns the r-th row of the matrix as vector.
1164	<code>init</code>	Initializes the matrix, setting its elements to the values passed to the constructor.
1163	<code>init_identity</code>	Initializes the matrix and sets its elements to the identity matrix.
1163	<code>init_zero</code>	Initializes the matrix and sets its elements to zero.
1165	<code>inverse</code>	Calculates the inverse of the matrix.
1164	<code>set_column</code>	Sets c-th column of the matrix with a vector.
1164	<code>set_row</code>	Sets r-th row of the matrix with a vector.
1165	<code>transpose</code>	Returns the transposition of the matrix.

64.10.3 Tmatrix3_single.init_zero

Synopsis: Initializes the matrix and sets its elements to zero.

Declaration: `constructor init_zero`

Visibility: default

64.10.4 Tmatrix3_single.init_identity

Synopsis: Initializes the matrix and sets its elements to the identity matrix.

Declaration: `constructor init_identity`

Visibility: default

Description: Initializes the matrix and sets its elements to the identity matrix, that is, elements to 1 on the left-upper to right-lower diagonal, the rest zero.

64.10.5 Tmatrix3_single.init

Synopsis: Initializes the matrix, setting its elements to the values passed to the constructor.

Declaration: `constructor init(aa: single; ab: single; ac: single; ba: single;
bb: single; bc: single; ca: single; cb: single;
cc: single)`

Visibility: default

Description: Initializes the matrix, setting its elements to the values passed to the constructor. The order of the values is left to right, then top to bottom.

64.10.6 Tmatrix3_single.get_column

Synopsis: Returns the c-th column of the matrix as vector.

Declaration: `function get_column(c: Byte) : Tvector3_single`

Visibility: default

Description: Returns the c-th column of the matrix as vector. The column numbering starts at 0.

64.10.7 Tmatrix3_single.get_row

Synopsis: Returns the r-th row of the matrix as vector.

Declaration: `function get_row(r: Byte) : Tvector3_single`

Visibility: default

Description: Returns the r-th row of the matrix as vector. The row numbering starts at 0.

64.10.8 Tmatrix3_single.set_column

Synopsis: Sets c-th column of the matrix with a vector.

Declaration: `procedure set_column(c: Byte; const v: Tvector3_single)`

Visibility: default

Description: Replaces the c-th column of the matrix with vector v. The column numbering starts at 0.

64.10.9 Tmatrix3_single.set_row

Synopsis: Sets r-th row of the matrix with a vector.

Declaration: `procedure set_row(r: Byte; const v: Tvector3_single)`

Visibility: default

Description: Replaces the r-th row of the matrix with vector v. The row numbering starts at 0.

64.10.10 Tmatrix3_single.determinant

Synopsis: Calculates the determinant of the matrix.

Declaration: `function determinant : single`

Visibility: default

Description: Returns the determinant of the matrix.

64.10.11 Tmatrix3_single.inverse

Synopsis: Calculates the inverse of the matrix.

Declaration: `function inverse (Adeterminant: single) : Tmatrix3_single`

Visibility: default

Description: `Tmatrix3_single.inverse` returns a new matrix that is the inverse of the matrix. You must pass the determinant of the matrix as parameter.

64.10.12 Tmatrix3_single.transpose

Synopsis: Returns the transposition of the matrix.

Declaration: `function transpose : Tmatrix3_single`

Visibility: default

Description: `Tmatrix2_single.transpose` returns a new matrix that is the transposition of the matrix, that is, the matrix with the x and y coordinates of the values swapped.

64.11 Tmatrix4_double**64.11.1 Description**

The `Tmatrix4_double` object provides a matrix of 4*4 double precision scalars.

64.11.2 Method overview

Page	Method	Description
1167	<code>determinant</code>	Calculates the determinant of the matrix.
1166	<code>get_column</code>	Returns the c-th column of the matrix as vector.
1166	<code>get_row</code>	Returns the r-th row of the matrix as vector.
1166	<code>init</code>	Initializes the matrix, setting its elements to the values passed to the constructor.
1166	<code>init_identity</code>	Initializes the matrix and sets its elements to the identity matrix.
1166	<code>init_zero</code>	Initializes the matrix and sets its elements to zero.
1167	<code>inverse</code>	Calculates the inverse of the matrix.
1167	<code>set_column</code>	Sets c-th column of the matrix with a vector.
1167	<code>set_row</code>	Sets r-th row of the matrix with a vector.
1167	<code>transpose</code>	Returns the transposition of the matrix.

64.11.3 Tmatrix4_double.init_zero

Synopsis: Initializes the matrix and sets its elements to zero.

Declaration: `constructor init_zero`

Visibility: default

64.11.4 Tmatrix4_double.init_identity

Synopsis: Initializes the matrix and sets its elements to the identity matrix.

Declaration: `constructor init_identity`

Visibility: default

Description: Initializes the matrix and sets its elements to the identity matrix, that is, elements to 1 on the left-upper to right-lower diagonal, the rest zero.

64.11.5 Tmatrix4_double.init

Synopsis: Initializes the matrix, setting its elements to the values passed to the constructor.

Declaration: `constructor init(aa: Double; ab: Double; ac: Double; ad: Double;
ba: Double; bb: Double; bc: Double; bd: Double;
ca: Double; cb: Double; cc: Double; cd: Double;
da: Double; db: Double; dc: Double; dd: Double)`

Visibility: default

Description: Initializes the matrix, setting its elements to the values passed to the constructor. The order of the values is left to right, then top to bottom.

64.11.6 Tmatrix4_double.get_column

Synopsis: Returns the c-th column of the matrix as vector.

Declaration: `function get_column(c: Byte) : Tvector4_double`

Visibility: default

Description: Returns the c-th column of the matrix as vector. The column numbering starts at 0.

64.11.7 Tmatrix4_double.get_row

Synopsis: Returns the r-th row of the matrix as vector.

Declaration: `function get_row(r: Byte) : Tvector4_double`

Visibility: default

Description: Returns the r-th row of the matrix as vector. The row numbering starts at 0.

64.11.8 Tmatrix4_double.set_column

Synopsis: Sets c-th column of the matrix with a vector.

Declaration: `procedure set_column(c: Byte; const v: Tvector4_double)`

Visibility: default

Description: Replaces the c-th column of the matrix with vector v. The column numbering starts at 0.

64.11.9 Tmatrix4_double.set_row

Synopsis: Sets r-th row of the matrix with a vector.

Declaration: `procedure set_row(r: Byte; const v: Tvector4_double)`

Visibility: default

Description: Replaces the r-th row of the matrix with vector v. The row numbering starts at 0.

64.11.10 Tmatrix4_double.determinant

Synopsis: Calculates the determinant of the matrix.

Declaration: `function determinant : Double`

Visibility: default

Description: Returns the determinant of the matrix. Note: Calculating the determinant of a 4*4 matrix requires quite a few operations.

64.11.11 Tmatrix4_double.inverse

Synopsis: Calculates the inverse of the matrix.

Declaration: `function inverse(Adeterminant: Double) : Tmatrix4_double`

Visibility: default

Description: `Tmatrix4_double.inverse` returns a new matrix that is the inverse of the matrix. You must pass the determinant of the matrix as parameter. Note: Calculating the inverse of a 4*4 matrix requires quite a few operations.

64.11.12 Tmatrix4_double.transpose

Synopsis: Returns the transposition of the matrix.

Declaration: `function transpose : Tmatrix4_double`

Visibility: default

Description: `Tmatrix2_double.transpose` returns a new matrix that is the transposition of the matrix, that is, the matrix with the x and y coordinates of the values swapped.

64.12 Tmatrix4_extended

64.12.1 Description

The Tmatrix4_extended object provides a matrix of 4*4 extended precision scalars.

64.12.2 Method overview

Page	Method	Description
1169	determinant	Calculates the determinant of the matrix.
1169	get_column	Returns the c-th column of the matrix as vector.
1169	get_row	Returns the r-th row of the matrix as vector.
1168	init	Initializes the matrix, setting its elements to the values passed to the constructor.
1168	init_identity	Initializes the matrix and sets its elements to the identity matrix.
1168	init_zero	Initializes the matrix and sets its elements to zero.
1170	inverse	Calculates the inverse of the matrix.
1169	set_column	Sets c-th column of the matrix with a vector.
1169	set_row	Sets r-th row of the matrix with a vector.
1170	transpose	Returns the transposition of the matrix.

64.12.3 Tmatrix4_extended.init_zero

Synopsis: Initializes the matrix and sets its elements to zero.

Declaration: `constructor init_zero`

Visibility: default

64.12.4 Tmatrix4_extended.init_identity

Synopsis: Initializes the matrix and sets its elements to the identity matrix.

Declaration: `constructor init_identity`

Visibility: default

Description: Initializes the matrix and sets its elements to the identity matrix, that is, elements to 1 on the left-upper to right-lower diagonal, the rest zero.

64.12.5 Tmatrix4_extended.init

Synopsis: Initializes the matrix, setting its elements to the values passed to the constructor.

Declaration: `constructor init(aa: extended; ab: extended; ac: extended;
ad: extended; ba: extended; bb: extended; bc: extended;
bd: extended; ca: extended; cb: extended; cc: extended;
cd: extended; da: extended; db: extended; dc: extended;
dd: extended)`

Visibility: default

Description: Initializes the matrix, setting its elements to the values passed to the constructor. The order of the values is left to right, then top to bottom.

64.12.6 Tmatrix4_extended.get_column

Synopsis: Returns the *c*-th column of the matrix as vector.

Declaration: `function get_column(c: Byte) : Tvector4_extended`

Visibility: default

Description: Returns the *c*-th column of the matrix as vector. The column numbering starts at 0.

64.12.7 Tmatrix4_extended.get_row

Synopsis: Returns the *r*-th row of the matrix as vector.

Declaration: `function get_row(r: Byte) : Tvector4_extended`

Visibility: default

Description: Returns the *r*-th row of the matrix as vector. The row numbering starts at 0.

64.12.8 Tmatrix4_extended.set_column

Synopsis: Sets *c*-th column of the matrix with a vector.

Declaration: `procedure set_column(c: Byte; const v: Tvector4_extended)`

Visibility: default

Description: Replaces the *c*-th column of the matrix with vector *v*. The column numbering starts at 0.

64.12.9 Tmatrix4_extended.set_row

Synopsis: Sets *r*-th row of the matrix with a vector.

Declaration: `procedure set_row(r: Byte; const v: Tvector4_extended)`

Visibility: default

Description: Replaces the *r*-th row of the matrix with vector *v*. The row numbering starts at 0.

64.12.10 Tmatrix4_extended.determinant

Synopsis: Calculates the determinant of the matrix.

Declaration: `function determinant : extended`

Visibility: default

Description: Returns the determinant of the matrix. Note: Calculating the determinant of a 4*4 matrix requires quite a few operations.

64.12.11 Tmatrix4_extended.inverse

Synopsis: Calculates the inverse of the matrix.

Declaration: `function inverse (Adeterminant: extended) : Tmatrix4_extended`

Visibility: default

Description: `Tmatrix4_extended.inverse` returns a new matrix that is the inverse of the matrix. You must pass the determinant of the matrix as parameter. Note: Calculating the inverse of a 4*4 matrix requires quite a few operations.

64.12.12 Tmatrix4_extended.transpose

Synopsis: Returns the transposition of the matrix.

Declaration: `function transpose : Tmatrix4_extended`

Visibility: default

Description: `Tmatrix2_extended.transpose` returns a new matrix that is the transposition of the matrix, that is, the matrix with the x and y coordinates of the values swapped.

64.13 Tmatrix4_single

64.13.1 Description

The `Tmatrix4_single` object provides a matrix of 4*4 single precision scalars.

64.13.2 Method overview

Page	Method	Description
1172	<code>determinant</code>	Calculates the determinant of the matrix.
1171	<code>get_column</code>	Returns the c-th column of the matrix as vector.
1171	<code>get_row</code>	Returns the r-th row of the matrix as vector.
1171	<code>init</code>	Initializes the matrix, setting its elements to the values passed to the constructor.
1171	<code>init_identity</code>	Initializes the matrix and sets its elements to the identity matrix.
1170	<code>init_zero</code>	Initializes the matrix and sets its elements to zero.
1172	<code>inverse</code>	Calculates the inverse of the matrix.
1171	<code>set_column</code>	Sets c-th column of the matrix with a vector.
1172	<code>set_row</code>	Sets r-th row of the matrix with a vector.
1172	<code>transpose</code>	Returns the transposition of the matrix.

64.13.3 Tmatrix4_single.init_zero

Synopsis: Initializes the matrix and sets its elements to zero.

Declaration: `constructor init_zero`

Visibility: default

64.13.4 Tmatrix4_single.init_identity

Synopsis: Initializes the matrix and sets its elements to the identity matrix.

Declaration: `constructor init_identity`

Visibility: default

Description: Initializes the matrix and sets its elements to the identity matrix, that is, elements to 1 on the left-upper to right-lower diagonal, the rest zero.

64.13.5 Tmatrix4_single.init

Synopsis: Initializes the matrix, setting its elements to the values passed to the constructor.

Declaration: `constructor init(aa: single; ab: single; ac: single; ad: single;
ba: single; bb: single; bc: single; bd: single;
ca: single; cb: single; cc: single; cd: single;
da: single; db: single; dc: single; dd: single)`

Visibility: default

Description: Initializes the matrix, setting its elements to the values passed to the constructor. The order of the values is left to right, then top to bottom.

64.13.6 Tmatrix4_single.get_column

Synopsis: Returns the c-th column of the matrix as vector.

Declaration: `function get_column(c: Byte) : Tvector4_single`

Visibility: default

Description: Returns the c-th column of the matrix as vector. The column numbering starts at 0.

64.13.7 Tmatrix4_single.get_row

Synopsis: Returns the r-th row of the matrix as vector.

Declaration: `function get_row(r: Byte) : Tvector4_single`

Visibility: default

Description: Returns the r-th row of the matrix as vector. The row numbering starts at 0.

64.13.8 Tmatrix4_single.set_column

Synopsis: Sets c-th column of the matrix with a vector.

Declaration: `procedure set_column(c: Byte; const v: Tvector4_single)`

Visibility: default

Description: Replaces the c-th column of the matrix with vector v. The column numbering starts at 0.

64.13.9 Tmatrix4_single.set_row

Synopsis: Sets r-th row of the matrix with a vector.

Declaration: `procedure set_row(r: Byte; const v: Tvector4_single)`

Visibility: default

Description: Replaces the r-th row of the matrix with vector v. The row numbering starts at 0.

64.13.10 Tmatrix4_single.determinant

Synopsis: Calculates the determinant of the matrix.

Declaration: `function determinant : single`

Visibility: default

Description: Returns the determinant of the matrix. Note: Calculating the determinant of a 4*4 matrix requires quite a few operations.

64.13.11 Tmatrix4_single.inverse

Synopsis: Calculates the inverse of the matrix.

Declaration: `function inverse(Adeterminant: single) : Tmatrix4_single`

Visibility: default

Description: `Tmatrix4_single.inverse` returns a new matrix that is the inverse of the matrix. You must pass the determinant of the matrix as parameter. Note: Calculating the inverse of a 4*4 matrix requires quite a few operations.

64.13.12 Tmatrix4_single.transpose

Synopsis: Returns the transposition of the matrix.

Declaration: `function transpose : Tmatrix4_single`

Visibility: default

Description: `Tmatrix2_single.transpose` returns a new matrix that is the transposition of the matrix, that is, the matrix with the x and y coordinates of the values swapped.

64.14 Tvector2_double

64.14.1 Description

The `Tvector2_double` object provides a vector of two double precision scalars.

64.14.2 Method overview

Page	Method	Description
1173	<code>init</code>	Initializes the vector, setting its elements to the values passed to the constructor.
1173	<code>init_one</code>	Initializes the vector and sets its elements to one.
1173	<code>init_zero</code>	Initializes the vector and sets its elements to zero.
1173	<code>length</code>	Calculates the length of the vector.
1173	<code>squared_length</code>	Calculates the squared length of the vector.

64.14.3 Tvector2_double.init_zero

Synopsis: Initializes the vector and sets its elements to zero.

Declaration: `constructor init_zero`

Visibility: default

64.14.4 Tvector2_double.init_one

Synopsis: Initializes the vector and sets its elements to one.

Declaration: `constructor init_one`

Visibility: default

64.14.5 Tvector2_double.init

Synopsis: Initializes the vector, setting its elements to the values passed to the constructor.

Declaration: `constructor init(a: Double; b: Double)`

Visibility: default

64.14.6 Tvector2_double.length

Synopsis: Calculates the length of the vector.

Declaration: `function &length : Double`

Visibility: default

Description: Calculate the length of the vector: `length=sqrt(data[0]**2+data[1]**2)`. Try to use `squared_length` ([1173](#)) if you are able to, as it is faster.

64.14.7 Tvector2_double.squared_length

Synopsis: Calculates the squared length of the vector.

Declaration: `function squared_length : Double`

Visibility: default

Description: Calculate the squared length of the vector: `squared_length=data[0]**2+data[1]**2`.

64.15 Tvector2_extended

64.15.1 Description

The `Tvector2_extended` object provides a vector of two extended precision scalars.

64.15.2 Method overview

Page	Method	Description
1174	<code>init</code>	Initializes the vector, setting its elements to the values passed to the constructor.
1174	<code>init_one</code>	Initializes the vector and sets its elements to one.
1174	<code>init_zero</code>	Initializes the vector and sets its elements to zero.
1174	<code>length</code>	Calculates the length of the vector.
1175	<code>squared_length</code>	Calculates the squared length of the vector.

64.15.3 Tvector2_extended.init_zero

Synopsis: Initializes the vector and sets its elements to zero.

Declaration: `constructor init_zero`

Visibility: default

64.15.4 Tvector2_extended.init_one

Synopsis: Initializes the vector and sets its elements to one.

Declaration: `constructor init_one`

Visibility: default

64.15.5 Tvector2_extended.init

Synopsis: Initializes the vector, setting its elements to the values passed to the constructor.

Declaration: `constructor init(a: extended; b: extended)`

Visibility: default

64.15.6 Tvector2_extended.length

Synopsis: Calculates the length of the vector.

Declaration: `function &length : extended`

Visibility: default

Description: Calculate the length of the vector: `length=sqrt(data[0]**2+data[1]**2)`. Try to use `squared_length` ([1175](#)) if you are able to, as it is faster.

64.15.7 Tvector2_extended.squared_length

Synopsis: Calculates the squared length of the vector.

Declaration: `function squared_length : extended`

Visibility: default

Description: Calculate the squared length of the vector: `squared_length=data[0]**2+data[1]**2`.

64.16 Tvector2_single

64.16.1 Description

The `Tvector2_single` object provides a vector of two single precision scalars.

64.16.2 Method overview

Page	Method	Description
1175	<code>init</code>	Initializes the vector, setting its elements to the values passed to the constructor.
1175	<code>init_one</code>	Initializes the vector and sets its elements to one.
1175	<code>init_zero</code>	Initializes the vector and sets its elements to zero.
1176	<code>length</code>	Calculates the length of the vector.
1176	<code>squared_length</code>	Calculates the squared length of the vector.

64.16.3 Tvector2_single.init_zero

Synopsis: Initializes the vector and sets its elements to zero.

Declaration: `constructor init_zero`

Visibility: default

64.16.4 Tvector2_single.init_one

Synopsis: Initializes the vector and sets its elements to one.

Declaration: `constructor init_one`

Visibility: default

64.16.5 Tvector2_single.init

Synopsis: Initializes the vector, setting its elements to the values passed to the constructor.

Declaration: `constructor init(a: single; b: single)`

Visibility: default

64.16.6 Tvector2_single.length

Synopsis: Calculates the length of the vector.

Declaration: `function &length : single`

Visibility: default

Description: Calculate the length of the vector: `length=sqrt(data[0]**2+data[1]**2)`. Try to use `squared_length` (1176) if you are able to, as it is faster.

64.16.7 Tvector2_single.squared_length

Synopsis: Calculates the squared length of the vector.

Declaration: `function squared_length : single`

Visibility: default

Description: Calculate the squared length of the vector: `squared_length=data[0]**2+data[1]**2`.

64.17 Tvector3_double

64.17.1 Description

The `Tvector3_double` object provides a vector of three double precision scalars.

64.17.2 Method overview

Page	Method	Description
1177	<code>init</code>	Initializes the vector, setting its elements to the values passed to the constructor.
1176	<code>init_one</code>	Initializes the vector and sets its elements to one.
1176	<code>init_zero</code>	Initializes the vector and sets its elements to zero.
1177	<code>length</code>	Calculates the length of the vector.
1177	<code>squared_length</code>	Calculates the squared length of the vector.

64.17.3 Tvector3_double.init_zero

Synopsis: Initializes the vector and sets its elements to zero.

Declaration: `constructor init_zero`

Visibility: default

64.17.4 Tvector3_double.init_one

Synopsis: Initializes the vector and sets its elements to one.

Declaration: `constructor init_one`

Visibility: default

64.17.5 Tvector3_double.init

Synopsis: Initializes the vector, setting its elements to the values passed to the constructor.

Declaration: `constructor init(a: Double; b: Double; c: Double)`

Visibility: default

64.17.6 Tvector3_double.length

Synopsis: Calculates the length of the vector.

Declaration: `function &length : Double`

Visibility: default

Description: Calculate the length of the vector: `length=sqrt(data[0]**2+data[1]**2+data[2]**2)`. Try to use `squared_length` ([1177](#)) if you are able to, as it is faster.

64.17.7 Tvector3_double.squared_length

Synopsis: Calculates the squared length of the vector.

Declaration: `function squared_length : Double`

Visibility: default

Description: Calculate the squared length of the vector: `squared_length=data[0]**2+data[1]**2+data[2]**2`.

64.18 Tvector3_extended

64.18.1 Description

The `Tvector3_extended` object provides a vector of three extended precision scalars.

64.18.2 Method overview

Page	Method	Description
1178	<code>init</code>	Initializes the vector, setting its elements to the values passed to the constructor.
1178	<code>init_one</code>	Initializes the vector and sets its elements to one.
1177	<code>init_zero</code>	Initializes the vector and sets its elements to zero.
1178	<code>length</code>	Calculates the length of the vector.
1178	<code>squared_length</code>	Calculates the squared length of the vector.

64.18.3 Tvector3_extended.init_zero

Synopsis: Initializes the vector and sets its elements to zero.

Declaration: `constructor init_zero`

Visibility: default

64.18.4 Tvector3_extended.init_one

Synopsis: Initializes the vector and sets its elements to one.

Declaration: `constructor init_one`

Visibility: default

64.18.5 Tvector3_extended.init

Synopsis: Initializes the vector, setting its elements to the values passed to the constructor.

Declaration: `constructor init(a: extended; b: extended; c: extended)`

Visibility: default

64.18.6 Tvector3_extended.length

Synopsis: Calculates the length of the vector.

Declaration: `function &length : extended`

Visibility: default

Description: Calculate the length of the vector: `length=sqrt(data[0]**2+data[1]**2+data[2]**2)`. Try to use `squared_length` (1178) if you are able to, as it is faster.

64.18.7 Tvector3_extended.squared_length

Synopsis: Calculates the squared length of the vector.

Declaration: `function squared_length : extended`

Visibility: default

Description: Calculate the squared length of the vector: `squared_length=data[0]**2+data[1]**2+data[2]**2`.

64.19 Tvector3_single

64.19.1 Description

The `Tvector3_single` object provides a vector of three single precision scalars.

64.19.2 Method overview

Page	Method	Description
1179	<code>init</code>	Initializes the vector, setting its elements to the values passed to the constructor.
1179	<code>init_one</code>	Initializes the vector and sets its elements to one.
1179	<code>init_zero</code>	Initializes the vector and sets its elements to zero.
1179	<code>length</code>	Calculates the length of the vector.
1179	<code>squared_length</code>	Calculates the squared length of the vector.

64.19.3 Tvector3_single.init_zero

Synopsis: Initializes the vector and sets its elements to zero.

Declaration: `constructor init_zero`

Visibility: default

64.19.4 Tvector3_single.init_one

Synopsis: Initializes the vector and sets its elements to one.

Declaration: `constructor init_one`

Visibility: default

64.19.5 Tvector3_single.init

Synopsis: Initializes the vector, setting its elements to the values passed to the constructor.

Declaration: `constructor init(a: single; b: single; c: single)`

Visibility: default

64.19.6 Tvector3_single.length

Synopsis: Calculates the length of the vector.

Declaration: `function &length : single`

Visibility: default

Description: Calculate the length of the vector: $\text{length} = \sqrt{\text{data}[0]**2 + \text{data}[1]**2 + \text{data}[2]**2}$. Try to use `squared_length` ([1179](#)) if you are able to, as it is faster.

64.19.7 Tvector3_single.squared_length

Synopsis: Calculates the squared length of the vector.

Declaration: `function squared_length : single`

Visibility: default

Description: Calculate the squared length of the vector: $\text{squared_length} = \text{data}[0]**2 + \text{data}[1]**2 + \text{data}[2]**2$.

64.20 Tvector4_double**64.20.1 Description**

The `Tvector4_double` object provides a vector of four double precision scalars.

64.20.2 Method overview

Page	Method	Description
1180	<code>init</code>	Initializes the vector, setting its elements to the values passed to the constructor.
1180	<code>init_one</code>	Initializes the vector and sets its elements to one.
1180	<code>init_zero</code>	Initializes the vector and sets its elements to zero.
1180	<code>length</code>	Calculates the length of the vector.
1180	<code>squared_length</code>	Calculates the squared length of the vector.

64.20.3 Tvector4_double.init_zero

Synopsis: Initializes the vector and sets its elements to zero.

Declaration: `constructor init_zero`

Visibility: default

64.20.4 Tvector4_double.init_one

Synopsis: Initializes the vector and sets its elements to one.

Declaration: `constructor init_one`

Visibility: default

64.20.5 Tvector4_double.init

Synopsis: Initializes the vector, setting its elements to the values passed to the constructor.

Declaration: `constructor init(a: Double; b: Double; c: Double; d: Double)`

Visibility: default

64.20.6 Tvector4_double.length

Synopsis: Calculates the length of the vector.

Declaration: `function &length : Double`

Visibility: default

Description: Calculate the length of the vector: `length=sqrt(data[0]**2+data[1]**2+data[2]**2+data[3]**2)`. Try to use `squared_length` ([1180](#)) if you are able to, as it is faster.

64.20.7 Tvector4_double.squared_length

Synopsis: Calculates the squared length of the vector.

Declaration: `function squared_length : Double`

Visibility: default

Description: Calculate the squared length of the vector: `squared_length=data[0]**2+data[1]**2+data[2]**2+data[3]**2`.

64.21 Tvector4_extended

64.21.1 Description

The `Tvector4_extended` object provides a vector of four extended precision scalars.

64.21.2 Method overview

Page	Method	Description
1181	<code>init</code>	Initializes the vector, setting its elements to the values passed to the constructor.
1181	<code>init_one</code>	Initializes the vector and sets its elements to one.
1181	<code>init_zero</code>	Initializes the vector and sets its elements to zero.
1181	<code>length</code>	Calculates the length of the vector.
1182	<code>squared_length</code>	Calculates the squared length of the vector.

64.21.3 Tvector4_extended.init_zero

Synopsis: Initializes the vector and sets its elements to zero.

Declaration: `constructor init_zero`

Visibility: default

64.21.4 Tvector4_extended.init_one

Synopsis: Initializes the vector and sets its elements to one.

Declaration: `constructor init_one`

Visibility: default

64.21.5 Tvector4_extended.init

Synopsis: Initializes the vector, setting its elements to the values passed to the constructor.

Declaration: `constructor init(a: extended; b: extended; c: extended; d: extended)`

Visibility: default

64.21.6 Tvector4_extended.length

Synopsis: Calculates the length of the vector.

Declaration: `function &length : extended`

Visibility: default

Description: Calculate the length of the vector: `length=sqrt(data[0]**2+data[1]**2+data[2]**2+data[3]**2)`. Try to use `squared_length` ([1182](#)) if you are able to, as it is faster.

64.21.7 Tvector4_extended.squared_length

Synopsis: Calculates the squared length of the vector.

Declaration: `function squared_length : extended`

Visibility: default

Description: Calculate the squared length of the vector: `squared_length=data[0]**2+data[1]**2+data[2]**2+data[3]**2`.

64.22 Tvector4_single

64.22.1 Description

The `Tvector4_single` object provides a vector of four single precision scalars.

64.22.2 Method overview

Page	Method	Description
1182	<code>init</code>	Initializes the vector, setting its elements to the values passed to the constructor.
1182	<code>init_one</code>	Initializes the vector and sets its elements to one.
1182	<code>init_zero</code>	Initializes the vector and sets its elements to zero.
1183	<code>length</code>	Calculates the length of the vector.
1183	<code>squared_length</code>	Calculates the squared length of the vector.

64.22.3 Tvector4_single.init_zero

Synopsis: Initializes the vector and sets its elements to zero.

Declaration: `constructor init_zero`

Visibility: default

64.22.4 Tvector4_single.init_one

Synopsis: Initializes the vector and sets its elements to one.

Declaration: `constructor init_one`

Visibility: default

64.22.5 Tvector4_single.init

Synopsis: Initializes the vector, setting its elements to the values passed to the constructor.

Declaration: `constructor init(a: single; b: single; c: single; d: single)`

Visibility: default

64.22.6 Tvector4_single.length

Synopsis: Calculates the length of the vector.

Declaration: `function &length : single`

Visibility: default

Description: Calculate the length of the vector: `length=sqrt(data[0]**2+data[1]**2+data[2]**2+data[3]**2)`. Try to use `squared_length` ([1183](#)) if you are able to, as it is faster.

64.22.7 Tvector4_single.squared_length

Synopsis: Calculates the squared length of the vector.

Declaration: `function squared_length : single`

Visibility: default

Description: Calculate the squared length of the vector: `squared_length=data[0]**2+data[1]**2+data[2]**2+data[3]**2`.

Chapter 65

Reference for unit 'mmx'

65.1 Used units

Table 65.1: Used units by unit 'mmx'

Name	Page
System	1335

65.2 Overview

This document describes the MMX unit. This unit allows you to use the MMX capabilities of the Free Pascal compiler. It was written by Florian Klaempfl for the I386 processor. It should work on all platforms that use the Intel processor.

65.3 Constants, types and variables

65.3.1 Constants

```
is_amd_3d_cpu : Boolean = False
```

The `is_amd_3d_cpu` initialized constant allows you to determine if the computer has the AMD 3D extensions. It is set correctly in the unit's initialization code.

```
is_amd_3d_dsp_cpu : Boolean = False
```

The `is_amd_3d_dsp_cpu` initialized constant allows you to determine if the computer has the AMD 3D DSP extensions. It is set correctly in the unit's initialization code.

```
is_amd_3d_mmx_cpu : Boolean = False
```

The `is_amd_3d_mmx_cpu` initialized constant allows you to determine if the computer has the AMD 3D MMX extensions. It is set correctly in the unit's initialization code.

```
is_mmx_cpu : Boolean = False
```

The `is_mmx_cpu` initialized constant allows you to determine if the computer has MMX extensions. It is set correctly in the unit's initialization code.

```
is_sse2_cpu : Boolean = False
```

The `is_sse2_cpu` initialized constant allows you to determine if the computer has the SSE2 extensions. It is set correctly in the unit's initialization code.

```
is_sse_cpu : Boolean = False
```

The `is_sse_cpu` initialized constant allows you to determine if the computer has the SSE extensions. It is set correctly in the unit's initialization code.

65.3.2 Types

```
pmmxbyte = ^tmmxbyte
```

Pointer to `tmmxbyte` (1185) array type.

```
pmmxcardinal = ^tmmxcardinal
```

Pointer to `tmmxcardinal` (1185) array type.

```
pmmxinteger = ^tmmxinteger
```

Pointer to `tmmxinteger` (1186) array type.

```
pmmxlongint = ^tmmxlongint
```

Pointer to `tmmxlongint` (1186) array type.

```
pmmxshortint = ^tmmxshortint
```

Pointer to `tmmxshortint` (1186) array type.

```
pmmxsingle = ^tmmxsingle
```

Pointer to `tmmxsingle` (1186) array type.

```
pmmxword = ^tmmxword
```

Pointer to `tmmxword` (1186) array type.

```
tmmxbyte = Array[0..7] of Byte
```

Array of bytes, 64 bits in size.

```
tmmxcardinal = Array[0..1] of Cardinal
```

Array of cardinals, 64 bits in size.

```
tmmxinteger = Array[0..3] of Integer
```

Array of integers, 64 bits in size.

```
tmmxlongint = Array[0..1] of LongInt
```

Array of longint, 64 bits in size.

```
tmmxshortint = Array[0..7] of ShortInt
```

Array of shortints, 64 bits in size.

```
tmmxsingle = Array[0..1] of single
```

Array of singles, 64 bits in size.

```
tmmxword = Array[0..3] of Word
```

Array of words, 64 bits in size.

65.4 Procedures and functions

65.4.1 emms

Synopsis: Reset floating point registers.

Declaration: `procedure emms`

Visibility: `default`

Description: `Emms` sets all floating point registers to empty. This procedure must be called after you have used any MMX instructions, if you want to use floating point arithmetic. If you just want to move floating point data around, it isn't necessary to call this function, the compiler doesn't use the FPU registers when moving data. Only when doing calculations, you should use this function. The following code demonstrates this:

```
Program MMXDemo;
uses mmx;
var
  d : double;
  a : array[0..10000] of double;
  i : longint;
begin
  d:=1.0;
{$mmx+}
  { floating point data is used, but we do _no_ arithmetic }
  for i:=0 to 10000 do
    a[i]:=d; { this is done with 64 bit moves }
{$mmx-}
  emms; { clear fpu }
  { now we can do floating point arithmetic again }
end.
```

See also: `femms` ([1187](#))

65.4.2 femms

Synopsis: Reset floating point registers - AMD version.

Declaration: `procedure femms`

Visibility: `default`

Description: `femms` executes the `femms` assembler instruction for AMD processors. it is not supported by all assemblers, hence it is coded as byte codes.

See also: `emms` ([1186](#))

Chapter 66

Reference for unit 'Mouse'

66.1 Used units

Table 66.1: Used units by unit 'Mouse'

Name	Page
System	1335

66.2 Overview

The `Mouse` unit implements a platform independent mouse handling interface. It is implemented identically on all platforms supported by Free Pascal and can be enhanced with custom drivers, should this be needed. It is intended to be used only in text-based screens, for instance in conjunction with the keyboard and video unit. No support for graphical screens is implemented, and there are (currently) no plans to implement this.

66.3 Writing a custom mouse driver.

The `mouse` unit has support for adding a custom mouse driver. This can be used to add support for mice not supported by the standard Free Pascal driver, but also to enhance an existing driver for instance to log mouse events or to implement a record and playback function.

The following unit shows how a mouse driver can be enhanced by adding some logging capabilities to the driver.

Listing: `./examples/mouseex/logmouse.pp`

```
unit logmouse;
```

```
interface
```

```
Procedure StartMouseLogging;
```

```
Procedure StopMouseLogging;
```

```
Function IsMouseLogging : Boolean;
```

```
Procedure SetMouseLogFileName(FileName : String);
```

implementation

uses sysutils,Mouse;

var

NewMouseDriver,
OldMouseDriver : TMouseDriver;
Active,Logging : Boolean;
LogFileName : **String**;
MouseLog : Text;

Function TimeStamp : **String**;

begin

TimeStamp:=**FormatDateTime**('hh:nn:ss',**Time**());
end;

Procedure StartMouseLogging;

begin

Logging:=True;
WriteLn(MouseLog,'Start logging mouse events at: ',TimeStamp);
end;

Procedure StopMouseLogging;

begin

WriteLn(MouseLog,'Stop logging mouse events at: ',TimeStamp);
Logging:=False;
end;

Function IsMouseLogging : Boolean;

begin

IsMouseLogging:=Logging;
end;

Procedure LogGetMouseEvent(**Var** Event : TMouseEvent);

Var

M : TMouseEvent;

begin

OldMouseDriver.GetMouseEvent(M);
If Logging **then**
 begin
 Write(MouseLog,TimeStamp,': Mouse ');
 With M **do**
 begin
 Case Action **of**
 MouseActionDown : **Write**(MouseLog,'down');
 MouseActionUp : **Write**(MouseLog,'up');
 MouseActionMove : **Write**(MouseLog,'move');
 end;
 Write(MouseLog,' event at ',X,',',Y);
 If (Buttons<>0) **then**
 begin

```

    Write(MouseLog,' for buttons: ');
    If (Buttons and MouseLeftbutton)<>0 then
        Write(MouseLog,'Left ');
    If (Buttons and MouseRightbutton)<>0 then
        Write(MouseLog,'Right ');
    If (Buttons and MouseMiddlebutton)<>0 then
        Write(MouseLog,'Middle ');
    end;
    Writeln(MouseLog);
end;
end;
end;

```

```

Procedure LogInitMouse;

```

```

begin
    OldMouseDriver.InitDriver();
    Assign(MouseLog,logFileName);
    Rewrite(MouseLog);
    Active:=True;
    StartMouseLogging;
end;

```

```

Procedure LogDoneMouse;

```

```

begin
    StopMouseLogging;
    Close(MouseLog);
    Active:=False;
    OldMouseDriver.DoneDriver();
end;

```

```

Procedure SetMouseLogFileName(FileName : String);

```

```

begin
    If Not Active then
        LogFileName:=FileName;
end;

```

```

Initialization

```

```

    GetMouseDriver(OldMouseDriver);
    NewMouseDriver:=OldMouseDriver;
    NewMouseDriver.GetMouseEvent:=@LogGetMouseEvent;
    NewMouseDriver.InitDriver:=@LogInitMouse;
    NewMouseDriver.DoneDriver:=@LogDoneMouse;
    LogFileName:='Mouse.log';
    Logging:=False;
    SetMouseDriver(NewMouseDriver);
end.

```

66.4 Constants, types and variables

66.4.1 Constants

```

errMouseBase = 1030

```

Base for mouse error codes.

```
errMouseInitError = errMouseBase + 0
```

Mouse initialization error.

```
errMouseNotImplemented = errMouseBase + 1
```

Mouse driver not implemented.

```
MouseActionDown = $0001
```

Mouse button down event signal.

```
MouseActionMove = $0004
```

Mouse cursor move event signal.

```
MouseActionUp = $0002
```

Mouse button up event signal.

```
MouseButton4 = $08
```

4th mouse button event.

```
MouseButton5 = $10
```

5th mouse button event.

```
MouseEventBufSize = 16
```

The mouse unit has a mechanism to buffer mouse events. This constant defines the size of the event buffer.

```
MouseLeftButton = $01
```

Left mouse button event.

```
MouseMiddleButton = $04
```

Middle mouse button event.

```
MouseRightButton = $02
```

Right mouse button event.

```
MouseXButton1 = $20
```

```
MouseXButton2 = $40
```


66.4.2 Types

`PMouseEvent = ^TMouseEvent`

Pointer to `TMouseEvent` (1199) record.

66.4.3 Variables

`MouseButtons : Byte`

This variable keeps track of the last known mouse button state. Do not use.

`MouseIntFlag : Byte`

This variable keeps track of the last known internal mouse state. Do not use.

`MouseWhereX : Word`

This variable keeps track of the last known cursor position. Do not use.

`MouseWhereY : Word`

This variable keeps track of the last known cursor position. Do not use.

66.5 Procedures and functions

66.5.1 DetectMouse

Synopsis: Detect the presence of a mouse.

Declaration: `function DetectMouse : Byte`

Visibility: default

Description: `DetectMouse` detects whether a mouse is attached to the system or not. If there is no mouse, then zero is returned. If a mouse is attached, then the number of mouse buttons is returned.

This function should be called after the mouse driver was initialized.

Errors: None.

See also: `InitMouse` (1196), `DoneMouse` (1193)

Listing: `./examples/mouseex/ex1.pp`

Program Example1;

{ Program to demonstrate the DetectMouse function. }

Uses mouse;

Var

Buttons : Byte;

begin

```

InitMouse;
Buttons:=DetectMouse;
if Buttons=0 then
  Writeln('No mouse present.')
else
  Writeln('Found mouse with ',Buttons,' buttons. ');
  DoneMouse;
end.

```

66.5.2 DoneMouse

Synopsis: Deinitialize mouse driver.

Declaration: `procedure DoneMouse`

Visibility: `default`

Description: `DoneMouse` De-initializes the mouse driver. It cleans up any memory allocated when the mouse was initialized, or removes possible mouse hooks from memory. The mouse functions will not work after `DoneMouse` was called. If `DoneMouse` is called a second time, it will exit at once. `InitMouse` should be called before `DoneMouse` can be called again.

For an example, see most other mouse functions.

Errors: None.

See also: `DetectMouse` ([1192](#)), `InitMouse` ([1196](#))

66.5.3 GetMouseButtons

Synopsis: Get the state of the mouse buttons.

Declaration: `function GetMouseButtons : Word`

Visibility: `default`

Description: `GetMouseButtons` returns the current button state of the mouse, i.e. it returns a or-ed combination of the following constants:

MouseLeftButton When the left mouse button is held down.

MouseRightButton When the right mouse button is held down.

MouseMiddleButton When the middle mouse button is held down.

Errors: None.

See also: `GetMouseEvent` ([1194](#)), `GetMouseX` ([1194](#)), `GetMouseY` ([1195](#))

Listing: `./examples/mouseex/ex2.pp`

Program `Example2;`

{ Program to demonstrate the GetMouseButtons function. }

Uses `mouse;`

begin
 `InitMouse;`

```

WriteLn('Press right mouse button to exit program');
While (GetMouseButtons<>MouseRightButton) do ;
  DoneMouse;
end.

```

66.5.4 GetMouseDriver

Synopsis: Get a copy of the currently active mouse driver.

Declaration: `procedure GetMouseDriver (var Driver: TMouseDriver)`

Visibility: default

Description: `GetMouseDriver` returns the currently set mouse driver. It can be used to retrieve the current mouse driver, and override certain callbacks.

A more detailed explanation about getting and setting mouse drivers can be found in [mousedrv \(1188\)](#).

For an example, see the section on writing a custom mouse driver, [mousedrv \(1188\)](#)

Errors: None.

See also: [SetMouseDriver \(1197\)](#)

66.5.5 GetMouseEvent

Synopsis: Get next mouse event from the queue.

Declaration: `procedure GetMouseEvent (var MouseEvent: TMouseEvent)`

Visibility: default

Description: `GetMouseEvent` returns the next mouse event (a movement, button press or button release), and waits for one if none is available in the queue.

Some mouse drivers can implement a mouse event queue which can hold multiple events till they are fetched. Others don't, and in that case, a one-event queue is implemented for use with `PollMouseEvent` ([1196](#)).

Errors: None.

See also: [GetMouseButtons \(1193\)](#), [GetMouseX \(1194\)](#), [GetMouseY \(1195\)](#)

66.5.6 GetMouseX

Synopsis: Query the current horizontal position of the mouse cursor.

Declaration: `function GetMouseX : Word`

Visibility: default

Description: `GetMouseX` returns the current X position of the mouse. X is measured in characters, starting at 0 for the left side of the screen.

Errors: None.

See also: [GetMouseButtons \(1193\)](#), [GetMouseEvent \(1194\)](#), [GetMouseY \(1195\)](#)

Listing: ./examples/mouseex/ex4.pp

Program Example4;

{ Program to demonstrate the GetMouseX,GetMouseY functions. }

Uses mouse;

Var

X,Y : Word;

begin

InitMouse;

WriteLn('Move mouse cursor to square 10,10 to end');

Repeat

X:=GetMouseX;

Y:=GetMouseY;

WriteLn('X,Y= (',X,',',Y,')');

Until (X=9) **and** (Y=9);

DoneMouse;

end.

66.5.7 GetMouseY

Synopsis: Query the current vertical position of the mouse cursor.

Declaration: `function GetMouseY : Word`

Visibility: default

Description: `GetMouseY` returns the current Y position of the mouse. Y is measured in characters, starting at 0 for the top of the screen.

For an example, see `GetMouseX` ([1194](#))

Errors: None.

See also: `GetMouseButtons` ([1193](#)), `GetMouseEvent` ([1194](#)), `GetMouseX` ([1194](#))

66.5.8 HideMouse

Synopsis: Hide the mouse cursor.

Declaration: `procedure HideMouse`

Visibility: default

Description: `HideMouse` hides the mouse cursor. This may or may not be implemented on all systems, and depends on the driver.

Errors: None.

See also: `ShowMouse` ([1198](#))

Listing: ./examples/mouseex/ex5.pp

Program Example5;*{ Program to demonstrate the HideMouse function. }***Uses** mouse;**Var**Event : TMouseEvent;
Visible: Boolean;**begin**InitMouse;
ShowMouse;
Visible:=True;
WriteLn('Press left mouse button to hide/show, right button quits');
Repeat
 GetMouseEvent(Event);
 With Event **do**
 If (Buttons=MouseLeftbutton) **and**
 (Action=MouseActionDown) **then**
 begin
 If Visible **then**
 HideMouse
 else
 ShowMouse;
 Visible:=**Not** Visible;
 end;
 Until (Event.Buttons=MouseRightButton) **and**
 (Event.Action=MouseActionDown);
 DoneMouse;
end.

66.5.9 InitMouse

Synopsis: Initialize the FPC mouse driver.

Declaration: `procedure InitMouse`

Visibility: default

Description: `InitMouse` Initializes the mouse driver. This will allocate any data structures needed for the mouse to function. All mouse functions can be used after a call to `InitMouse`.A call to `InitMouse` must always be followed by a call to `DoneMouse` ([1193](#)) at program exit. Failing to do so may leave the mouse in an unusable state, or may result in memory leaks.

For an example, see most other functions.

Errors: None.

See also: `DoneMouse` ([1193](#)), `DetectMouse` ([1192](#))**66.5.10 PollMouseEvent**

Synopsis: Query next mouse event. Do not wait if none available.

Declaration: `function PollMouseEvent (var MouseEvent : TMouseEvent) : Boolean`

Visibility: default

Description: `PollMouseEvent` checks whether a mouse event is available, and returns it in `MouseEvent` if one is found. The function result is `True` in that case. If no mouse event is pending, the function result is `False`, and the contents of `MouseEvent` is undefined.

Note that after a call to `PollMouseEvent`, the event should still be removed from the mouse event queue with a call to `GetMouseEvent`.

Errors: None.

See also: `GetMouseEvent` (1194), `PutMouseEvent` (1197)

66.5.11 PutMouseEvent

Synopsis: Put a mouse event in the event queue.

Declaration: `procedure PutMouseEvent(const MouseEvent: TMouseEvent)`

Visibility: default

Description: `PutMouseEvent` adds `MouseEvent` to the input queue. The next call to `GetMouseEvent` (1194) or `PollMouseEvent` will then return `MouseEvent`.

Please note that depending on the implementation the mouse event queue can hold only one value.

Errors: None.

See also: `GetMouseEvent` (1194), `PollMouseEvent` (1196)

66.5.12 SetMouseDriver

Synopsis: Set a new mouse driver.

Declaration: `procedure SetMouseDriver(const Driver: TMouseDriver)`

Visibility: default

Description: `SetMouseDriver` sets the mouse driver to `Driver`. This function should be called before `InitMouse` (1196) is called, or after `DoneMouse` is called. If it is called after the mouse has been initialized, it does nothing.

For more information on setting the mouse driver, `mousedrv` (1188).

For an example, see `mousedrv` (1188)

See also: `InitMouse` (1196), `DoneMouse` (1193), `GetMouseDriver` (1194)

66.5.13 SetMouseXY

Synopsis: Set the mouse cursor position.

Declaration: `procedure SetMouseXY(x: Word; y: Word)`

Visibility: default

Description: `SetMouseXY` places the mouse cursor on `X`, `Y`. `X` and `Y` are zero based character coordinates: 0, 0 is the top-left corner of the screen, and the position is in character cells (i.e. not in pixels).

Errors: None.

See also: [GetMouseX \(1194\)](#), [GetMouseY \(1195\)](#)

Listing: ./examples/mouseex/ex7.pp

Program Example7;

{ Program to demonstrate the SetMouseXY function. }

Uses mouse;

begin

 InitMouse;

WriteLn('Click right mouse button to quit.');

 SetMouseXY(40,12);

Repeat

WriteLn(GetMouseX,',',GetMouseY);

If (GetMouseX>70) **then**

 SetMouseXY(10,GetMouseY);

If (GetMouseY>20) **then**

 SetMouseXY(GetMouseX,5);

Until (GetMouseButtons=MouseRightButton);

 DoneMouse;

end.

66.5.14 ShowMouse

Synopsis: Show the mouse cursor.

Declaration: `procedure ShowMouse`

Visibility: default

Description: `ShowMouse` shows the mouse cursor if it was previously hidden. The capability to hide or show the mouse cursor depends on the driver.

For an example, see [HideMouse \(1195\)](#)

Errors: None.

See also: [HideMouse \(1195\)](#)

66.6 TMouseDriver

```
TMouseDriver = record
  UseDefaultQueue : Boolean;
  InitDriver :
  procedure;
  DoneDriver : procedure;
  DetectMouse : function : Byte
  ;
  ShowMouse : procedure;
  HideMouse : procedure;
  GetMouseX :
  function : Word;
  GetMouseY : function : Word;
```

```
GetMouseButtons
: function : Word;
SetMouseXY : procedure(x: Word; y: Word);
GetMouseEvent : procedure(var MouseEvent: TMouseEvent);
PollMouseEvent
: function(var MouseEvent: TMouseEvent) : Boolean;
PutMouseEvent
: procedure(const MouseEvent: TMouseEvent);
end
```

The `TMouseDriver` record is used to implement a mouse driver in the `SetMouseDriver` (1197) function. Its fields must be filled in before calling the `SetMouseDriver` (1197) function.

66.7 TMouseEvent

```
TMouseEvent = packed record
  buttons : Word;
  x : Word;
  y : Word
;
  Action : Word;
end
```

The `TMouseEvent` is the central type of the mouse unit, it is used to describe all mouse events.

The `Buttons` field describes which buttons were down when the event occurred. The `x`, `y` fields describe where the event occurred on the screen. The `Action` describes what action was going on when the event occurred. The `Buttons` and `Action` field can be examined using the constants defined in the unit interface.

Chapter 67

Reference for unit 'Objects'

67.1 Used units

Table 67.1: Used units by unit 'Objects'

Name	Page
System	1335

67.2 Overview

This document documents the `objects` unit. The unit was implemented by many people, and was mainly taken from the FreeVision sources. It has been ported to all supported platforms.

The methods and fields that are in a `Private` part of an object declaration have been left out of this documentation.

67.3 Constants, types and variables

67.3.1 Constants

`coIndexError = - 1`

Collection list error: Index out of range.

`coOverflow = - 2`

Collection list error: Overflow.

`DefaultTPCompatible : Boolean = False`

`DefaultTPCompatible` is used to initialize `tstream.tpcompatible` (??).

`MaxBytes = 128 * 1024 * 128`

Maximum data size (in bytes).

```
MaxCollectionSize = MaxBytes div SizeOf(Pointer)
```

Maximum collection size (in items).

```
MaxPtrs = MaxBytes div SizeOf(Pointer)
```

Maximum data size (in pointers).

```
MaxReadBytes = $7fffffff
```

Maximum data that can be read from a stream (not used).

```
MaxTPCompatibleCollectionSize = 65520 div 4
```

Maximum collection size (in items, same value as in TP).

```
MaxWords = MaxBytes div SizeOf(Word)
```

Maximum data size (in words).

```
RCollection : TStreamRec = (ObjType: 50; VmtLink: Ofs(^ TypeOf(TCollection
)); Load: @ TCollection.Load; Store: @ TCollection.Store; Next: Nil
)
```

Default stream record for the TCollection ([1217](#)) object.

```
RStrCollection : TStreamRec = (ObjType: 69; VmtLink: Ofs(^ TypeOf
(TStrCollection)); Load: @ TStrCollection.Load; Store: @ TStrCollection
.Store; Next: Nil)
```

Default stream record for the TStrCollection ([1257](#)) object.

```
RStringCollection : TStreamRec = (ObjType: 51; VmtLink: Ofs(^ TypeOf
(TStringCollection)); Load: @ TStringCollection.Load; Store: @ TStringCollection
.Store; Next: Nil)
```

Default stream record for the TStringCollection ([1267](#)) object.

```
RStringList : TStreamRec = (ObjType: 52; VmtLink: Ofs(^ TypeOf(TStringList
)); Load: @ TStringList.Load; Store: Nil; Next: Nil)
```

Default stream record for the TStringList ([1269](#)) object.

```
RStrListMaker : TStreamRec = (ObjType: 52; VmtLink: Ofs(^ TypeOf(TStrListMaker
)); Load: Nil; Store: @ TStrListMaker.Store; Next: Nil)
```

Default stream record for the TStrListMaker ([1271](#)) object.

```
stCreate = $3C00
```

Stream initialization mode: Create new file.

`stError = - 1`

Stream error codes: Access error.

`stGetError = - 5`

Stream error codes: Get object error.

`stInitError = - 2`

Stream error codes: Initialize error.

`stOk = 0`

Stream error codes: No error.

`stOpen = $3D02`

Stream initialization mode: Read/write access.

`stOpenError = - 8`

Stream error codes: Error opening stream.

`stOpenRead = $3D00`

Stream initialization mode: Read access only.

`stOpenWrite = $3D01`

Stream initialization mode: Write access only.

`stPutError = - 6`

Stream error codes: Put object error.

`stReadError = - 3`

Stream error codes: Stream read error.

`StreamError : CodePointer = Nil`

Pointer to default stream error handler.

`stSeekError = - 7`

Stream error codes: Seek error in stream.

`stWriteError = - 4`

Stream error codes: Stream write error.

`vmtHeaderSize = 8`

Size of the VMT header in an object (not used).

67.3.2 Types

`AsciiZ = Array[0..255] of Char`

Filename - null terminated array of characters.

`FNameStr = String`

Filename - shortstring version.

`PBufStream = ^TBufStream`

Pointer to `TBufStream` ([1213](#)) object.

`PByteArray = ^TByteArray`

Pointer to `TByteArray` ([1205](#)).

`PCharSet = ^TCharSet`

Pointer to `TCharSet` ([1205](#)).

`PCollection = ^TCollection`

Pointer to `TCollection` ([1217](#)) object.

`PDosStream = ^TDosStream`

Pointer to `TDosStream` ([1232](#)) object.

`PItemList = ^TItemList`

Pointer to `TItemList` ([1205](#)) object.

`PMemoryStream = ^TMemoryStream`

Pointer to `TMemoryStream` ([1237](#)) object.

`PObject = ^TObject`

Pointer to `TObject` ([1239](#)) object.

`PPoint = ^TPoint`

Pointer to `TPoint` ([1241](#)) record.

`PPointerArray = ^TPointerArray`

Pointer to `TPointerArray` ([1205](#)).

`PRect = ^TRect`

Pointer to TRect (1241) object.

PResourceCollection = ^TResourceCollection

Pointer to TResourceCollection (1247) object.

PResourceFile = ^TResourceFile

Pointer to TResourceFile (1248) object.

PSortedCollection = ^TSortedCollection

Pointer to TSortedCollection (1251) object.

PStrCollection = ^TStrCollection

Pointer to TStrCollection (1257) object.

PStream = ^TStream

Pointer type to TStream (1259).

PStreamRec = ^TStreamRec

Pointer to TStreamRec (1213).

PStrIndex = ^TStrIndex

Pointer to TStrIndex (1205) array.

PString = PShortString

Pointer to a shortstring.

PStringCollection = ^TStringCollection

Pointer to TStringCollection (1267) object.

PStringList = ^TStringList

Pointer to TStringList (1269) object.

PStrListMaker = ^TStrListMaker

Pointer to TStrListMaker (1271) object.

PUnSortedStrCollection = ^TUnSortedStrCollection

Pointer to TUnSortedStrCollection (1272) object.

PWordArray = ^TWordArray

Pointer to TWordArray ([1205](#)).

`Sw_Integer = LongInt`

Alias for longint.

`Sw_Word = Cardinal`

Alias for Cardinal.

`TByteArray = Array[0..MaxBytes-1] of Byte`

Array with maximum allowed number of bytes.

`TCharSet = Set of Char`

Generic set of characters type.

`TItemList = Array[0..MaxCollectionSize-1] of Pointer`

Pointer array type used in a TCollection ([1217](#)).

`TPointerArray = Array[0..MaxPtrs-1] of Pointer`

Array with maximum allowed number of pointers.

`TStrIndex = Array[0..9999] of TStrIndexRec`

Pointer array type used in a TStringList ([1269](#)).

`TWordArray = Array[0..MaxWords-1] of Word`

Array with maximum allowed number of words.

67.3.3 Variables

`invalidhandle : THandle`

Value for invalid handle. Initial value for file stream handles or when the stream is closed.

67.4 Procedures and functions

67.4.1 Abstract

Synopsis: Abstract error handler.

Declaration: `procedure Abstract`

Visibility: default

Description: When implementing abstract methods, do not declare them as `abstract`. Instead, define them simply as `virtual`. In the implementation of such abstract methods, call the `Abstract` procedure. This allows explicit control of what happens when an abstract method is called.

The current implementation of `Abstract` terminates the program with a run-time error 211.

Errors: None.

67.4.2 CallPointerConstructor

Synopsis: Call a constructor with a pointer argument.

Declaration: `function CallPointerConstructor(Ctor: CodePointer; Obj: pointer;
VMT: pointer; Param1: pointer) : pointer`

Visibility: default

Description: `CallVoidConstructor` calls the constructor of an object. `Ctor` is the address of the constructor, `Obj` is a pointer to the instance. If it is `Nil`, then a new instance is allocated. `VMT` is a pointer to the object's VMT. `Param1` is passed to the constructor. The return value is a pointer to the instance.

Note that this can only be used on constructors that require a pointer as the sole argument. It can also be used to call a constructor with a single argument by reference.

Errors: If the constructor expects other arguments than a pointer, the stack may be corrupted.

See also: `CallVoidConstructor` (1207), `CallPointerMethod` (1206), `CallVoidLocal` (1207), `CallPointerLocal` (1206), `CallVoidMethodLocal` (1208), `CallPointerMethodLocal` (1207)

67.4.3 CallPointerLocal

Synopsis: Call a local nested function with a pointer argument.

Declaration: `function CallPointerLocal(Func: CodePointer; Frame: Pointer;
Param1: pointer) : pointer`

Visibility: default

Description: `CallPointerLocal` calls the local procedure with address `Func`, where `Frame` is the frame of the wrapping function. It passes `Param1` to the local function.

Errors: If the local function expects other parameters than a pointer, the stack may become corrupted.

See also: `CallPointerMethod` (1206), `CallVoidMethod` (1208), `CallVoidLocal` (1207), `CallVoidMethodLocal` (1208), `CallPointerMethodLocal` (1207), `CallVoidConstructor` (1207), `CallPointerConstructor` (1206)

67.4.4 CallPointerMethod

Synopsis: Call a method with a single pointer argument.

Declaration: `function CallPointerMethod(Method: CodePointer; Obj: pointer;
Param1: pointer) : pointer`

Visibility: default

Description: `CallPointerMethod` calls the method with address `Method` for instance `Obj`. It passes `Param1` to the method as the single argument. It returns a pointer to the instance.

Errors: If the method expects other parameters than a single pointer, the stack may become corrupted.

See also: `CallVoidMethod` (1208), `CallVoidLocal` (1207), `CallPointerLocal` (1206), `CallVoidMethodLocal` (1208), `CallPointerMethodLocal` (1207), `CallVoidConstructor` (1207), `CallPointerConstructor` (1206)

67.4.5 CallPointerMethodLocal

Synopsis: Call a local procedure of a method with a pointer argument.

Declaration: `function CallPointerMethodLocal(Func: CodePointer; Frame: Pointer;
Obj: pointer; Param1: pointer) : pointer`

Visibility: default

Description: `CallPointerMethodLocal` calls the local procedure with address `Func`, where `Frame` is the frame of the wrapping method. It passes `Param1` to the local function.

Errors: If the local function expects other parameters than a pointer, the stack may become corrupted.

See also: `CallPointerMethod` (1206), `CallVoidMethod` (1208), `CallPointerLocal` (1206), `CallVoidLocal` (1207), `CallVoidMethodLocal` (1208), `CallVoidConstructor` (1207), `CallPointerConstructor` (1206)

67.4.6 CallVoidConstructor

Synopsis: Call a constructor with no arguments.

Declaration: `function CallVoidConstructor(Ctor: CodePointer; Obj: pointer;
VMT: pointer) : pointer`

Visibility: default

Description: `CallVoidConstructor` calls the constructor of an object. `Ctor` is the address of the constructor, `Obj` is a pointer to the instance. If it is `Nil`, then a new instance is allocated. `VMT` is a pointer to the object's VMT. The return value is a pointer to the instance.

Note that this can only be used on constructors that require no arguments.

Errors: If the constructor expects arguments, the stack may be corrupted.

See also: `CallPointerConstructor` (1206), `CallPointerMethod` (1206), `CallVoidLocal` (1207), `CallPointerLocal` (1206), `CallVoidMethodLocal` (1208), `CallPointerMethodLocal` (1207)

67.4.7 CallVoidLocal

Synopsis: Call a local nested procedure.

Declaration: `function CallVoidLocal(Func: CodePointer; Frame: Pointer) : pointer`

Visibility: default

Description: `CallVoidLocal` calls the local procedure with address `Func`, where `Frame` is the frame of the wrapping function.

Errors: If the local function expects parameters, the stack may become corrupted.

See also: `CallPointerMethod` (1206), `CallVoidMethod` (1208), `CallPointerLocal` (1206), `CallVoidMethodLocal` (1208), `CallPointerMethodLocal` (1207), `CallVoidConstructor` (1207), `CallPointerConstructor` (1206)

67.4.8 CallVoidMethod

Synopsis: Call an object method.

Declaration: `function CallVoidMethod(Method: CodePointer; Obj: pointer) : pointer`

Visibility: default

Description: `CallVoidMethod` calls the method with address `Method` for instance `Obj`. It returns a pointer to the instance.

Errors: If the method expects parameters, the stack may become corrupted.

See also: `CallPointerMethod` (1206), `CallVoidLocal` (1207), `CallPointerLocal` (1206), `CallVoidMethodLocal` (1208), `CallPointerMethodLocal` (1207), `CallVoidConstructor` (1207), `CallPointerConstructor` (1206)

67.4.9 CallVoidMethodLocal

Synopsis: Call a local procedure of a method.

Declaration: `function CallVoidMethodLocal(Func: CodePointer; Frame: Pointer; Obj: pointer) : pointer`

Visibility: default

Description: `CallVoidMethodLocal` calls the local procedure with address `Func`, where `Frame` is the frame of the wrapping method.

Errors: If the local function expects parameters, the stack may become corrupted.

See also: `CallPointerMethod` (1206), `CallVoidMethod` (1208), `CallPointerLocal` (1206), `CallVoidLocal` (1207), `CallPointerMethodLocal` (1207), `CallVoidConstructor` (1207), `CallPointerConstructor` (1206)

67.4.10 DisposeStr

Synopsis: Dispose of a shortstring which was allocated on the heap.

Declaration: `procedure DisposeStr(P: PString)`

Visibility: default

Description: `DisposeStr` removes a dynamically allocated string from the heap.

For an example, see `NewStr` (1209).

Errors: None.

See also: `NewStr` (1209), `SetStr` (1212)

67.4.11 LongDiv

Synopsis: Overflow safe divide.

Declaration: `function LongDiv(X: LongInt; Y: Integer) : Integer`

Visibility: default

Description: `LongDiv` divides `X` by `Y`. The result is of type `Integer` instead of type `Longint`, as you would get normally.

Errors: If Y is zero, a run-time error will be generated.

See also: LongMul ([1209](#))

67.4.12 LongMul

Synopsis: Overflow safe multiply.

Declaration: `function LongMul(X: Integer; Y: Integer) : LongInt`

Visibility: default

Description: LongMul multiplies X with Y. The result is of type LongInt. This avoids possible overflow errors you would normally get when multiplying X and Y that are too big.

Errors: None.

See also: LongDiv ([1208](#))

67.4.13 NewStr

Synopsis: Allocate a copy of a shortstring on the heap.

Declaration: `function NewStr(const S: string) : PString`

Visibility: default

Description: NewStr makes a copy of the string S on the heap, and returns a pointer to this copy. If the string is empty then Nil is returned.

The allocated memory is not based on the declared size of the string passed to NewStr, but is based on the actual length of the string.

Errors: If not enough memory is available, an 'out of memory' error will occur.

See also: DisposeStr ([1208](#)), SetStr ([1212](#))

Listing: ./examples/objectex/ex40.pp

Program ex40;

{ Program to demonstrate the NewStr function }

Uses Objects;

Var S : **String**;
P : PString;

begin
S:='Some really cute string';
P:=**NewStr**(S);
If P^<>S **then**
 WriteLn ('Oh-oh... Something is wrong !!');
 DisposeStr(P);
end.

67.4.14 RegisterObjects

Synopsis: Register standard objects.

Declaration: `procedure RegisterObjects`

Visibility: `default`

Description: `RegisterObjects` registers the following objects for streaming:

1. `TCollection`, see `TCollection` (1217).
2. `TStringCollection`, see `TStringCollection` (1267).
3. `TStrCollection`, see `TStrCollection` (1257).

Errors: None.

See also: `RegisterType` (1210)

67.4.15 RegisterType

Synopsis: Register new object for streaming.

Declaration: `procedure RegisterType (var S: TStreamRec)`

Visibility: `default`

Description: `RegisterType` registers a new type for streaming. An object cannot be streamed unless it has been registered first. The stream record `S` needs to have the following fields set:

ObjType: Sw_Word This should be a unique identifier. Each possible type should have it's own identifier.

VmtLink: pointer This should contain a pointer to the VMT (Virtual Method Table) of the object you try to register.

Load : Pointer is a pointer to a method that initializes an instance of that object, and reads the initial values from a stream. This method should accept as it's sole argument a `PStream` type variable.

Store: Pointer is a pointer to a method that stores an instance of the object to a stream. This method should accept as it's sole argument a `PStream` type variable.

The VMT of the object can be retrieved with the following expression:

```
VmtLink: ofs (TypeOf (MyType) ^) ;
```

Errors: In case of error (if a object with the same `ObjType`) is already registered), run-time error 212 occurs.

Listing: `./examples/objectex/myobject.pp`

Unit `MyObject`;

Interface

Uses `Objects`;

Type

```

PMyObject = ^TMyObject;
TMyObject = Object(TObject)
  Field : Longint;
  Constructor Init;
  Constructor Load (Var Stream : TStream);
  Destructor Done;
  Procedure Store (Var Stream : TStream);
  Function GetField : Longint;
  Procedure SetField (Value : Longint);
end;

```

Implementation

Constructor TMyobject.Init;

```

begin
  Inherited Init;
  Field := -1;
end;

```

Constructor TMyobject.Load (Var Stream : TStream);

```

begin
  Stream.Read(Field, Sizeof(Field));
end;

```

Destructor TMyObject.Done;

```

begin
end;

```

Function TMyObject.GetField : Longint;

```

begin
  GetField := Field;
end;

```

Procedure TMyObject.SetField (Value : Longint);

```

begin
  Field := Value;
end;

```

Procedure TMyObject.Store (Var Stream : TStream);

```

begin
  Stream.Write(Field, SizeOf(Field));
end;

```

```

Const MyObjectRec : TStreamRec = (
  Objtype : 666;
  vmtlink : Ofs(TypeOf(TMyObject)^);
  Load : @TMyObject.Load;
  Store : @TMyObject.Store;
);

```

```

begin
  RegisterObjects;

```

```
RegisterType (MyObjectRec);
end.
```

67.4.16 SetStr

Synopsis: Allocate a copy of a shortstring on the heap.

Declaration: `procedure SetStr(var p: PString; const s: string)`

Visibility: default

Description: `SetStr` makes a copy of the string `S` on the heap and returns the pointer to this copy in `P`. If `P` pointed to another string (i.e. was not `Nil`, the memory is released first. Contrary to `NewStr` ([1209](#)), if the string is empty then a pointer to an empty string is returned.

The allocated memory is not based on the declared size of the string passed to `NewStr`, but is based on the actual length of the string.

Errors: If not enough memory is available, an 'out of memory' error will occur.

See also: `DisposeStr` ([1208](#)), `NewStr` ([1209](#))

67.5 LongRec

```
LongRec = packed record
  Hi : Word;
  Lo : Word;
end
```

Record describing a longint (in Words).

67.6 PtrRec

```
PtrRec = packed record
  Ofs : Word;
  Seg : Word;
end
```

Record describing a pointer to a memory location.

67.7 TStreamRec

```
TStreamRec = packed record
  ObjType : Sw_Word;
  VmtLink : pointer
;
  Load : CodePointer;
  Store : CodePointer;
```

```
    Next : PStreamRec
;
end
```

TStreamRec is used by the `Objects` unit streaming mechanism: when an object is registered, a TStreamRec record is added to a list of records. This list is used when objects need to be streamed from/streamed to a stream. It contains all the information needed to stream the object.

67.8 TStrIndexRec

```
TStrIndexRec = packed record
    Key : Sw_Word;
    Count : Word;
    Offset
    : Word;
end
```

Record type used in a TStringList ([1269](#)) to store the strings.

67.9 WordRec

```
WordRec = packed record
    Hi : Byte;
    Lo : Byte;
end
```

Record describing a Word (in bytes).

67.10 TBufStream

67.10.1 Description

Bufstream implements a buffered file stream. That is, all data written to the stream is written to memory first. Only when the buffer is full, or on explicit request, the data is written to disk.

Also, when reading from the stream, first the buffer is checked if there is any unread data in it. If so, this is read first. If not the buffer is filled again, and then the data is read from the buffer.

The size of the buffer is fixed and is set when constructing the file.

This is useful if you need heavy throughput for your stream, because it speeds up operations.

67.10.2 Method overview

Page	Method	Description
1215	Close	Flush data and Close the file.
1214	Done	Close the file and cleans up the instance.
1215	Flush	FLush data from buffer, and write it to stream.
1214	Init	Initialize an instance of <code>TBufStream</code> and open the file.
1216	Open	Open the file if it is closed.
1217	Read	Read data from the file to a buffer in memory.
1216	Seek	Set current position in file.
1216	Truncate	Flush buffer, and truncate the file at current position.
1217	Write	Write data to the file from a buffer in memory.

67.10.3 TBufStream.Init

Synopsis: Initialize an instance of `TBufStream` and open the file.

Declaration: `constructor Init (FileName: FNameStr; Mode: Word; Size: Word)`

Visibility: default

Description: `Init` instantiates an instance of `TBufStream`. The name of the file that contains (or will contain) the data of the stream is given in `FileName`. The `Mode` parameter determines whether a new file should be created and what access rights you have on the file. It can be one of the following constants:

stCreate Creates a new file.

stOpenRead Read access only.

stOpenWrite Write access only.

stOpen Read and write access.

The `Size` parameter determines the size of the buffer that will be created. It should be different from zero.

For an example see `TBufStream.Flush` ([1215](#)).

Errors: On error, `Status` is set to `stInitError`, and `ErrorInfo` is set to the dos error code.

See also: `TDosStream.Init` ([1233](#)), `TBufStream.Done` ([1214](#))

67.10.4 TBufStream.Done

Synopsis: Close the file and cleans up the instance.

Declaration: `destructor Done; Virtual`

Visibility: default

Description: `Done` flushes and closes the file if it was open and cleans up the instance of `TBufStream`.

For an example see `TBufStream.Flush` ([1215](#)).

Errors: None.

See also: `TDosStream.Done` ([1233](#)), `TBufStream.Init` ([1214](#)), `TBufStream.Close` ([1215](#))

67.10.5 TBufStream.Close

Synopsis: Flush data and Close the file.

Declaration: `procedure Close; Virtual`

Visibility: default

Description: `Close` flushes and closes the file if it was open, and sets `Handle` to -1. Contrary to `Done` (1214) it does not clean up the instance of `TBufStream`

For an example see `TBufStream.Flush` (1215).

Errors: None.

See also: `TStream.Close` (1263), `TBufStream.Init` (1214), `TBufStream.Done` (1214)

67.10.6 TBufStream.Flush

Synopsis: FLush data from buffer, and write it to stream.

Declaration: `procedure Flush; Virtual`

Visibility: default

Description: When the stream is in write mode, the contents of the buffer are written to disk, and the buffer position is set to zero. When the stream is in read mode, the buffer position is set to zero.

Errors: Write errors may occur if the file was in write mode. see `Write` (1217) for more info on the errors.

See also: `TStream.Close` (1263), `TBufStream.Init` (1214), `TBufStream.Done` (1214)

Listing: `./examples/objectex/ex15.pp`

Program `ex15`;

{ Program to demonstrate the TStream.Flush method }

Uses `Objects`;

Var `L : String`;

`P : PString`;

`S : PBufStream`; *{ Only one with Flush implemented. }*

begin

`L := 'Some constant string';`

{ Buffer size of 100 }

`S := New(PBufStream, Init('test.dat', stcreate, 100));`

Writeln ('Writing "', `L`, '" to stream with handle ', `S^.Handle`);

`S^.WriteStr(@L)`;

{ At this moment, there is no data on disk yet. }

`S^.Flush`;

{ Now there is. }

`S^.WriteStr(@L)`;

{ Close calls flush first }

`S^.Close`;

Writeln ('Closed stream. File handle is ', `S^.Handle`);

`S^.Open(stOpenRead)`;

`P := S^.ReadStr`;

`L := P^`;

```

DisposeStr(P);
WriteLn ('Read "',L,'" from stream with handle ',S^.Handle);
S^.Close;
Dispose (S,Done);
end.

```

67.10.7 TBufStream.Truncate

Synopsis: Flush buffer, and truncate the file at current position.

Declaration: `procedure Truncate; Virtual`

Visibility: default

Description: If the status of the stream is `stOK`, then `Truncate` tries to flush the buffer, and then truncates the stream size to the current file position.

For an example, see `TDosStream.Truncate` ([1234](#)).

Errors: Errors can be those of `Flush` ([1215](#)) or `TDosStream.Truncate` ([1234](#)).

See also: `TStream.Truncate` ([1264](#)), `TDosStream.Truncate` ([1234](#)), `TStream.GetSize` ([1261](#))

67.10.8 TBufStream.Seek

Synopsis: Set current position in file.

Declaration: `procedure Seek(Pos: LongInt); Virtual`

Visibility: default

Description: If the stream's status is `stOK`, then `Seek` sets the file position to `Pos`. `Pos` is a zero-based offset, counted from the beginning of the file.

For an example, see `TStream.Seek` ([1265](#));

Errors: In case an error occurs, the stream's status is set to `stSeekError`, and the OS error code is stored in `ErrorInfo`.

See also: `TStream.Seek` ([1265](#)), `TStream.GetPos` ([1261](#))

67.10.9 TBufStream.Open

Synopsis: Open the file if it is closed.

Declaration: `procedure Open(OpenMode: Word); Virtual`

Visibility: default

Description: If the stream's status is `stOK`, and the stream is closed then `Open` re-opens the file stream with mode `OpenMode`. This call can be used after a `Close` ([1215](#)) call.

For an example, see `TDosStream.Open` ([1235](#)).

Errors: If an error occurs when re-opening the file, then `Status` is set to `stOpenError`, and the OS error code is stored in `ErrorInfo`

See also: `TStream.Open` ([1263](#)), `TBufStream.Close` ([1215](#))

67.10.10 TBufStream.Read

Synopsis: Read data from the file to a buffer in memory.

Declaration: `procedure Read(var Buf; Count: LongInt); Virtual`

Visibility: default

Description: If the Stream is open and the stream status is `stOK` then `Read` will read `Count` bytes from the stream and place them in `Buf`.

`Read` will first try to read the data from the stream's internal buffer. If insufficient data is available, the buffer will be filled before continuing to read. This process is repeated until all needed data has been read.

For an example, see `TStream.Read` ([1266](#)).

Errors: In case of an error, `Status` is set to `StReadError`, and `ErrorInfo` gets the OS specific error, or 0 when an attempt was made to read beyond the end of the stream.

See also: `TStream.Read` ([1266](#)), `TBufStream.Write` ([1217](#))

67.10.11 TBufStream.Write

Synopsis: Write data to the file from a buffer in memory.

Declaration: `procedure Write(var Buf; Count: LongInt); Virtual`

Visibility: default

Description: If the Stream is open and the stream status is `stOK` then `Write` will write `Count` bytes from `Buf` and place them in the stream.

`Write` will first try to write the data to the stream's internal buffer. When the internal buffer is full, then the contents will be written to disk. This process is repeated until all data has been written.

For an example, see `TStream.Read` ([1266](#)).

Errors: In case of an error, `Status` is set to `StWriteError`, and `ErrorInfo` gets the OS specific error.

See also: `TStream.Write` ([1266](#)), `TBufStream.Read` ([1217](#))

67.11 TCollection

67.11.1 Description

The `TCollection` object manages a collection of pointers or objects. It also provides a series of methods to manipulate these pointers or objects.

Whether or not objects are used depends on the kind of calls you use. All kinds come in 2 flavors, one for objects, one for pointers.

67.11.2 Method overview

Page	Method	Description
1219	<code>At</code>	Return the item at a certain index.
1228	<code>AtDelete</code>	Delete item at certain position.
1227	<code>AtFree</code>	Free an item at the indicates position, calling it's destructor.
1231	<code>AtInsert</code>	Insert an element at a certain position in the collection.
1230	<code>AtPut</code>	Set collection item, overwriting an existing value.
1226	<code>Delete</code>	Delete an item from the collection, but does not destroy it.
1225	<code>DeleteAll</code>	Delete all elements from the collection. Objects are not destroyed.
1219	<code>Done</code>	Clean up collection, release all memory.
1230	<code>Error</code>	Set error code.
1222	<code>FirstThat</code>	Return first item which matches a test.
1229	<code>ForEach</code>	Execute procedure for each item in the list.
1225	<code>Free</code>	Free item from collection, calling it's destructor.
1224	<code>FreeAll</code>	Release all objects from the collection.
1228	<code>FreeItem</code>	Destroy a non-nil item.
1221	<code>GetItem</code>	Read one item off the stream.
1220	<code>IndexOf</code>	Find the position of a certain item.
1218	<code>Init</code>	Instantiate a new collection.
1226	<code>Insert</code>	Insert a new item in the collection at the end.
1221	<code>LastThat</code>	Return last item which matches a test.
1218	<code>Load</code>	Initialize a new collection and load collection from a stream.
1223	<code>Pack</code>	Remove all <code>>Nil</code> pointers from the collection.
1232	<code>PutItem</code>	Put one item on the stream.
1230	<code>SetLimit</code>	Set maximum number of elements in the collection.
1232	<code>Store</code>	Write collection to a stream.

67.11.3 TCollection.Init

Synopsis: Instantiate a new collection.

Declaration: `constructor Init (ALimit: Sw_Integer; ADelta: Sw_Integer)`

Visibility: default

Description: `Init` initializes a new instance of a collection. It sets the (initial) maximum number of items in the collection to `ALimit`. `ADelta` is the increase size : The number of memory places that will be allocated in case `ALimit` is reached, and another element is added to the collection.

For an example, see `TCollection.ForEach` ([1229](#)).

Errors: None.

See also: `TCollection.Load` ([1218](#)), `TCollection.Done` ([1219](#))

67.11.4 TCollection.Load

Synopsis: Initialize a new collection and load collection from a stream.

Declaration: `constructor Load (var S: TStream)`

Visibility: default

Description: `Load` initializes a new instance of a collection. It reads from stream `S` the item count, the item limit count, and the increase size. After that, it reads the specified number of items from the stream.

Errors: Errors returned can be those of `GetItem` ([1221](#)).

See also: `TCollection.Init` ([1218](#)), `TCollection.GetItem` ([1221](#)), `TCollection.Done` ([1219](#))

Listing: `./examples/objectex/ex22.pp`

Program `ex22`;

{ Program to demonstrate the TCollection.Load method }

Uses `Objects, MyObject`; *{ For TMyObject definition and registration }*

Var `C` : `PCollection`;
 `M` : `PMMyObject`;
 `I` : `Longint`;
 `S` : `PMemoryStream`;

begin
 `C:=New(PCollection, Init(100,10));`
 For `I:=1 to 100 do`
 begin
 `M:=New(PMyObject, Init);`
 `M^.SetField(100-I);`
 `C^.Insert(M);`
 end;
 WriteLn ('Inserted ', `C^.Count`, ' objects');
 `S:=New(PMemoryStream, Init(1000,10));`
 `C^.Store(S^);`
 `C^.FreeAll;`
 // Dispose(C, Done);
 `S^.Seek(0);`
 `C^.Load(S^);`
 WriteLn ('Read ', `C^.Count`, ' objects from stream.');

Dispose(`S`, `Done`);
 Dispose(`C`, `Done`);
end.

67.11.5 TCollection.Done

Synopsis: Clean up collection, release all memory.

Declaration: `destructor Done; Virtual`

Visibility: `default`

Description: `Done` frees all objects in the collection, and then releases all memory occupied by the instance.

For an example, see `TCollection.ForEach` ([1229](#)).

Errors: None.

See also: `TCollection.Init` ([1218](#)), `TCollection.FreeAll` ([1224](#))

67.11.6 TCollection.At

Synopsis: Return the item at a certain index.

Declaration: `function At(Index: Sw_Integer) : Pointer`

Visibility: default

Description: `At` returns the item at position `Index`.

Errors: If `Index` is less than zero or larger than the number of items in the collection, see `TCollection.Error` is called with `coIndexError` and `Index` as arguments, resulting in a run-time error.

See also: `TCollection.Insert` ([1226](#))

Listing: `./examples/objectex/ex23.pp`

Program `ex23`;

{ Program to demonstrate the TCollection.At method }

Uses `Objects, MyObject`; *{ For TMyObject definition and registration }*

Var `C` : `PCollection`;
 `M` : `PMMyObject`;
 `I` : `Longint`;

```
begin
  C:=New(PCollection,Init(100,10));
  For I:=1 to 100 do
    begin
      M:=New(PMyObject,Init);
      M^.SetField(100-I);
      C^.Insert(M);
    end;
  For I:=0 to C^.Count-1 do
    begin
      M:=C^.At(I);
      Writeln ('Object ',i,' has field : ',M^.GetField);
    end;
  C^.FreeAll;
  Dispose(C,Done);
end.
```

67.11.7 TCollection.IndexOf

Synopsis: Find the position of a certain item.

Declaration: `function IndexOf(Item: Pointer) : Sw_Integer; Virtual`

Visibility: default

Description: `IndexOf` returns the index of `Item` in the collection. If `Item` isn't present in the collection, -1 is returned.

Errors: If the item is not present, -1 is returned.

See also: `TCollection.At` ([1219](#)), `TCollection.GetItem` ([1221](#)), `TCollection.Insert` ([1226](#))

Listing: `./examples/objectex/ex24.pp`

Program `ex24`;

{ Program to demonstrate the TCollection.IndexOf method }

Uses Objects, MyObject; { For TMyObject definition and registration }

```

Var C : PCollection;
      M, Keep : PMyObject;
      I : Longint;

begin
  Randomize;
  C:=New(PCollection, Init(100,10));
  Keep:=Nil;
  For I:=1 to 100 do
    begin
      M:=New(PMyObject, Init);
      M^.SetField(I-1);
      If Random<0.1 then
        Keep:=M;
        C^.Insert(M);
      end;
    If Keep=Nil then
      begin
        WriteLn ('Please run again. No object selected');
        Halt(1);
      end;
    WriteLn ('Selected object has field : ', Keep^.GetField);
    Write ('Selected object has index : ', C^.IndexOf(Keep));
    WriteLn (' should match it's field. ');
    C^.FreeAll;
    Dispose(C, Done);
  end.

```

67.11.8 TCollection.GetItem

Synopsis: Read one item off the stream.

Declaration: `function GetItem(var S: TStream) : Pointer; Virtual`

Visibility: default

Description: `GetItem` reads a single item off the stream `S`, and returns a pointer to this item. This method is used internally by the `Load` method, and should not be used directly.

Errors: Possible errors are the ones from `TStream.Get` ([1259](#)).

See also: `TStream.Get` ([1259](#)), `TCollection.Store` ([1232](#))

67.11.9 TCollection.LastThat

Synopsis: Return last item which matches a test.

Declaration: `function LastThat(Test: CodePointer) : Pointer`

Visibility: default

Description: This function returns the last item in the collection for which `Test` returns a non-nil result. `Test` is a function that accepts 1 argument: a pointer to an object, and that returns a pointer as a result.

Errors: None.

See also: `TCollection.FirstThat` ([1222](#))

Listing: `./examples/objectex/ex25.pp`

Program `ex21`;

{ Program to demonstrate the TCollection.Foreach method }

Uses `Objects, MyObject`; *{ For TMyObject definition and registration }*

Var `C` : `PCollection`;
 `M` : `PMMyObject`;
 `I` : `Longint`;

Function `CheckField` (`Dummy`: `Pointer`; `P` : `PMMyObject`) : `Longint`;

begin

If `P^.GetField<56` **then**
 `Checkfield:=1`

else
 `CheckField:=0`;

end;

begin

`C:=New(PCollection, Init(100,10))`;

For `I:=1` **to** `100` **do**

begin

`M:=New(PMyObject, Init)`;

`M^.SetField(I)`;

`C^.Insert(M)`;

end;

Writeln ('Inserted ', `C^.Count`, ' objects');

Writeln ('Last one for which Field<56 has index (should be 54) : ',
 `C^.IndexOf(C^.LastThat(@CheckField))`);

`C^.FreeAll`;

Dispose(`C`, `Done`);

end.

67.11.10 TCollection.FirstThat

Synopsis: Return first item which matches a test.

Declaration: `function FirstThat (Test: CodePointer) : Pointer`

Visibility: default

Description: This function returns the first item in the collection for which `Test` returns a non-nil result. `Test` is a function that accepts 1 argument: a pointer to an object, and that returns a pointer as a result.

Errors: None.

See also: `TCollection.LastThat` ([1221](#))

Listing: `./examples/objectex/ex26.pp`

Program `ex21`;

{ Program to demonstrate the TCollection.FirstThat method }

Uses Objects,MyObject; { For TMyObject definition and registration }

Var C : PCollection;
 M : PMyObject;
 I : Longint;

Function CheckField (Dummy: Pointer;P : PMyObject) : Longint;

begin
 if P^.GetField>56 **then**
 Checkfield:=1
 else
 CheckField:=0;
end;

begin
 C:=**New**(PCollection,Init(100,10));
 For I:=1 **to** 100 **do**
 begin
 M:=**New**(PMyObject,Init);
 M^.SetField(I);
 C^.Insert(M);
 end;
 Writeln ('Inserted ',C^.Count,' objects');
 Writeln ('first one for which Field>56 has index (should be 56) : ',
 C^.IndexOf(C^.FirstThat(@CheckField)));
 C^.FreeAll;
 Dispose(C,Done);
end.

67.11.11 TCollection.Pack

Synopsis: Remove all >Nil pointers from the collection.

Declaration: `procedure Pack`

Visibility: default

Description: Pack removes all Nil pointers from the collection, and adjusts Count to reflect this change. No memory is freed as a result of this call. In order to free any memory, you can call SetLimit with an argument of Count after a call to Pack.

Errors: None.

See also: TCollection.SetLimit ([1230](#))

Listing: ./examples/objectex/ex26.pp

Program ex21;

{ Program to demonstrate the TCollection.FirstThat method }

Uses Objects,MyObject; { For TMyObject definition and registration }

Var C : PCollection;
 M : PMyObject;
 I : Longint;

```

Function CheckField (Dummy: Pointer;P : PMyObject) : Longint;

begin
  If P^.GetField>56 then
    Checkfield:=1
  else
    CheckField:=0;
end;

begin
  C:=New(PCollection,Init(100,10));
  For I:=1 to 100 do
    begin
      M:=New(PMyObject,Init);
      M^.SetField(I);
      C^.Insert(M);
    end;
  Writeln ('Inserted ',C^.Count,' objects');
  Writeln ('first one for which Field>56 has index (should be 56) : ',
    C^.IndexOf(C^.FirstThat(@CheckField)));
  C^.FreeAll;
  Dispose(C,Done);
end.

```

67.11.12 TCollection.FreeAll

Synopsis: Release all objects from the collection.

Declaration: `procedure FreeAll`

Visibility: default

Description: `FreeAll` calls the destructor of each object in the collection. It doesn't release any memory occupied by the collection itself, but it does set `Count` to zero.

See also: `TCollection.DeleteAll` ([1225](#)), `TCollection.FreeItem` ([1228](#))

Listing: `./examples/objectex/ex28.pp`

Program ex28;

{ Program to demonstrate the TCollection.FreeAll method }

Uses Objects,MyObject; *{ For TMyObject definition and registration }*

```

Var C : PCollection;
      M : PMyObject;
      I : Longint;

```

```

begin
  Randomize;
  C:=New(PCollection,Init(120,10));
  For I:=1 to 100 do
    begin
      M:=New(PMyObject,Init);
      M^.SetField(I-1);
      C^.Insert(M);
    end;

```

```

    end;
    Writeln ('Added 100 Items.');
```

```

    C^.FreeAll;
    Writeln ('Freed all objects.');
```

```

    Dispose(C,Done);
end.
```

67.11.13 TCollection.DeleteAll

Synopsis: Delete all elements from the collection. Objects are not destroyed.

Declaration: `procedure DeleteAll`

Visibility: `default`

Description: `DeleteAll` deletes all elements from the collection. It just sets the `Count` variable to zero. Contrary to `FreeAll` ([1224](#)), `DeleteAll` doesn't call the destructor of the objects.

Errors: None.

See also: `TCollection.FreeAll` ([1224](#)), `TCollection.Delete` ([1226](#))

Listing: `./examples/objectex/ex29.pp`

Program `ex29`;

```

{
  Program to demonstrate the TCollection.DeleteAll method
  Compare with example 28, where FreeAll is used.
}
```

Uses `Objects,MyObject`; { For *TMyObject* definition and registration }

```

Var C : PCollection;
      M : PMyObject;
      I : Longint;
```

```

begin
  Randomize;
  C:=New(PCollection,Init(120,10));
  For I:=1 to 100 do
    begin
      M:=New(PMyObject,Init);
      M^.SetField(I-1);
      C^.Insert(M);
    end;
  Writeln ('Added 100 Items.');
```

```

  C^.DeleteAll;
  Writeln ('Deleted all objects.');
```

```

  Dispose(C,Done);
end.
```

67.11.14 TCollection.Free

Synopsis: Free item from collection, calling it's destructor.

Declaration: `procedure Free(Item: Pointer)`

Visibility: `default`

Description: `Free` Deletes `Item` from the collection, and calls the destructor `Done` of the object.

Errors: If the `Item` is not in the collection, `Error` will be called with `coIndexError`.

See also: `TCollection.FreeItem` ([1228](#))

Listing: `./examples/objectex/ex30.pp`

Program `ex30`;

{ Program to demonstrate the TCollection.Free method }

Uses `Objects, MyObject`; *{ For TMyObject definition and registration }*

Var `C` : `PCollection`;
 `M` : `PMyObject`;
 `I` : `Longint`;

begin
 Randomize;
 `C:=New(PCollection, Init(120,10));`
 For `I:=1 to 100 do`
 begin
 `M:=New(PMyObject, Init);`
 `M^.SetField(I-1);`
 `C^.Insert(M);`
 end;
 Writeln ('Added 100 Items.');

With `C^ do`
 While `Count>0 do Free(At(Count-1));`
 Writeln ('Freed all objects.');

Dispose(`C, Done`);
end.

67.11.15 TCollection.Insert

Synopsis: Insert a new item in the collection at the end.

Declaration: `procedure Insert(Item: Pointer); Virtual`

Visibility: `default`

Description: `Insert` inserts `Item` in the collection. `TCollection` inserts this item at the end, but descendent objects may insert it at another place.

Errors: None.

See also: `TCollection.AtInsert` ([1231](#)), `TCollection.AtPut` ([1230](#))

67.11.16 TCollection.Delete

Synopsis: Delete an item from the collection, but does not destroy it.

Declaration: `procedure Delete(Item: Pointer)`

Visibility: default

Description: `Delete` deletes `Item` from the collection. It doesn't call the item's destructor, though. For this the `Free` (1225) call is provided.

Errors: If the `Item` is not in the collection, `Error` will be called with `coIndexError`.

See also: `TCollection.AtDelete` (1228), `TCollection.Free` (1225)

Listing: ./examples/objectex/ex31.pp

Program ex31;

{ Program to demonstrate the TCollection.Delete method }

Uses Objects, MyObject; *{ For TMyObject definition and registration }*

Var C : PCollection;
 M : PMyObject;
 I : Longint;

begin
 Randomize;
 C:=**New**(PCollection, Init(120,10));
 For I:=1 **to** 100 **do**
 begin
 M:=**New**(PMyObject, Init);
 M^.SetField(I-1);
 C^.Insert(M);
 end;
 Writeln ('Added 100 Items.');

With C^ **do**
 While Count>0 **do** **Delete**(At(Count-1));
 Writeln ('Freed all objects');

Dispose(C, Done);
end.

67.11.17 TCollection.AtFree

Synopsis: Free an item at the indicates position, calling it's destructor.

Declaration: `procedure AtFree (Index: Sw_Integer)`

Visibility: default

Description: `AtFree` deletes the item at position `Index` in the collection, and calls the item's destructor if it is not `Nil`.

Errors: If `Index` isn't valid then `Error` (1230) is called with `CoIndexError`.

See also: `TCollection.Free` (1225), `TCollection.AtDelete` (1228)

Listing: ./examples/objectex/ex32.pp

Program ex32;

{ Program to demonstrate the TCollection.AtFree method }

Uses Objects,MyObject; { For TMyObject definition and registration }

```

Var C : PCollection;
      M : PMyObject;
      I : Longint;

begin
  Randomize;
  C:=New(PCollection,Init(120,10));
  For I:=1 to 100 do
    begin
      M:=New(PMyObject,Init);
      M^.SetField(I-1);
      C^.Insert(M);
    end;
  Writeln ('Added 100 Items');
  With C^ do
    While Count>0 do AtFree(Count-1);
  Writeln ('Freed all objects. ');
  Dispose(C,Done);
end.

```

67.11.18 TCollection.FreeItem

Synopsis: Destroy a non-nil item.

Declaration: `procedure FreeItem(Item: Pointer); Virtual`

Visibility: default

Description: `FreeItem` calls the destructor of `Item` if it is not nil.

Remark This function is used internally by the `TCollection` object, and should not be called directly.

Errors: None.

See also: `TCollection.Free` ([1225](#)), `TCollection.AtFree` ([1227](#))

67.11.19 TCollection.AtDelete

Synopsis: Delete item at certain position.

Declaration: `procedure AtDelete(Index: Sw_Integer)`

Visibility: default

Description: `AtDelete` deletes the pointer at position `Index` in the collection. It doesn't call the object's destructor.

Errors: If `Index` isn't valid then `Error` ([1230](#)) is called with `CoIndexError`.

See also: `TCollection.Delete` ([1226](#))

Listing: `./examples/objectex/ex33.pp`

Program ex33;

{ Program to demonstrate the TCollection.AtDelete method }

Uses Objects,MyObject; *{ For TMyObject definition and registration }*

Var C : PCollection;
 M : PMyObject;
 I : Longint;

begin
 Randomize;
 C:=**New**(PCollection,Init(120,10));
 For I:=1 **to** 100 **do**
 begin
 M:=**New**(PMyObject,Init);
 M^.SetField(I-1);
 C^.Insert(M);
 end;
 Writeln ('Added 100 Items.');

With C^ **do**
 While Count>0 **do** AtDelete(Count-1);
 Writeln ('Freed all objects.');

Dispose(C,Done);
end.

67.11.20 TCollection.ForEach

Synopsis: Execute procedure for each item in the list.

Declaration: `procedure ForEach(Action: CodePointer)`

Visibility: default

Description: ForEach calls Action for each element in the collection, and passes the element as an argument to Action.

Action is a procedural type variable that accepts a pointer as an argument.

Errors: None.

See also: TCollection.FirstThat ([1222](#)), TCollection.LastThat ([1221](#))

Listing: ./examples/objectex/ex21.pp

Program ex21;

{ Program to demonstrate the TCollection.ForEach method }

Uses Objects,MyObject; *{ For TMyObject definition and registration }*

Var C : PCollection;
 M : PMyObject;
 I : Longint;

Procedure PrintField (Dummy: Pointer;P : PMyObject);

begin

```

    Writeln ('Field : ',P^.GetField);
end;

begin
    C:=New(PCollection,Init(100,10));
    For I:=1 to 100 do
        begin
            M:=New(PMyObject,Init);
            M^.SetField(100-I);
            C^.Insert(M);
        end;
    Writeln ('Inserted ',C^.Count,' objects');
    C^.ForEach(@PrintField);
    C^.FreeAll;
    Dispose(C,Done);
end.

```

67.11.21 TCollection.SetLimit

Synopsis: Set maximum number of elements in the collection.

Declaration: `procedure SetLimit (ALimit: Sw_Integer); Virtual`

Visibility: default

Description: `SetLimit` sets the maximum number of elements in the collection. `ALimit` must not be less than `Count`, and should not be larger than `MaxCollectionSize`

For an example, see Pack ([1223](#)).

Errors: None.

See also: `TCollection.Init` ([1218](#))

67.11.22 TCollection.Error

Synopsis: Set error code.

Declaration: `procedure Error (Code: Integer; Info: Integer); Virtual`

Visibility: default

Description: `Error` is called by the various `TCollection` methods in case of an error condition. The default behaviour is to make a call to `RunError` with an error of `212-Code`.

This method can be overridden by descendent objects to implement a different error-handling.

See also: `Abstract` ([1205](#))

67.11.23 TCollection.AtPut

Synopsis: Set collection item, overwriting an existing value.

Declaration: `procedure AtPut (Index: Sw_Integer; Item: Pointer)`

Visibility: default

Description: `AtPut` sets the element at position `Index` in the collection to `Item`. Any previous value is overwritten.

For an example, see `Pack` ([1223](#)).

Errors: If `Index` isn't valid then `Error` ([1230](#)) is called with `CoIndexError`.

67.11.24 TCollection.AtInsert

Synopsis: Insert an element at a certain position in the collection.

Declaration: `procedure AtInsert (Index: Sw_Integer; Item: Pointer)`

Visibility: default

Description: `AtInsert` inserts `Item` in the collection at position `Index`, shifting all elements by one position. In case the current limit is reached, the collection will try to expand with a call to `SetLimit`

Errors: If `Index` isn't valid then `Error` ([1230](#)) is called with `CoIndexError`. If the collection fails to expand, then `coOverflow` is passed to `Error`.

See also: `TCollection.Insert` ([1226](#))

Listing: `./examples/objectex/ex34.pp`

Program `ex34`;

{ Program to demonstrate the TCollection.AtInsert method }

Uses `Objects, MyObject`; *{ For TMyObject definition and registration }*

Var `C` : `PCollection`;
 `M` : `PMMyObject`;
 `I` : `Longint`;

Procedure `PrintField` (`Dummy`: `Pointer`; `P` : `PMMyObject`);

begin
 `Writeln` ('Field : ', `P^.GetField`);
end;

begin
 `Randomize`;
 `C:=New`(`PCollection`, `Init`(120,10));
 `Writeln` ('Inserting 100 records at random places.');

For `I:=1` **to** 100 **do**
 begin
 `M:=New`(`PMMyObject`, `Init`);
 `M^.SetField`(`I-1`);
 If `I=1` **then**
 `C^.Insert`(`M`)
 else
 With `C^` **do**
 `AtInsert`(`Random`(`Count`), `M`);
 end;
 `Writeln` ('Values : ');
 `C^.Foreach`(`@PrintField`);
 `Dispose`(`C`, `Done`);
 end.
end.

67.11.25 TCollection.Store

Synopsis: Write collection to a stream.

Declaration: `procedure Store(var S: TStream)`

Visibility: default

Description: `Store` writes the collection to the stream `S`. It does this by writing the current `Count`, `Limit` and `Delta` to the stream, and then writing each item to the stream.

The contents of the stream are then suitable for instantiating another collection with `Load` ([1218](#)).

For an example, see `TCollection.Load` ([1218](#)).

Errors: Errors returned are those by `TStream.Put` ([1264](#)).

See also: `TCollection.Load` ([1218](#)), `TCollection.PutItem` ([1232](#))

67.11.26 TCollection.PutItem

Synopsis: Put one item on the stream.

Declaration: `procedure PutItem(var S: TStream; Item: Pointer); Virtual`

Visibility: default

Description: `PutItem` writes `Item` to stream `S`. This method is used internally by the `TCollection` object, and should not be called directly.

Errors: Errors are those returned by `TStream.Put` ([1264](#)).

See also: `Store` ([1232](#)), `GetItem` ([1221](#))

67.12 TDosStream

67.12.1 Description

`TDosStream` is a stream that stores its contents in a file. it overrides a couple of methods of `TStream` ([1259](#)) for this.

In addition to the fields inherited from `TStream` (see `TStream` ([1259](#))), there are some extra fields, that describe the file. (mainly the name and the OS file handle)

No buffering in memory is done when using `TDosStream`. All data are written directly to the file. For a stream that buffers in memory, see `TBufStream` ([1213](#)).

67.12.2 Method overview

Page	Method	Description
1233	<code>Close</code>	Close the file.
1233	<code>Done</code>	Closes the file and cleans up the instance.
1233	<code>Init</code>	Instantiate a new instance of <code>TDosStream</code> .
1235	<code>Open</code>	Open the file stream.
1236	<code>Read</code>	Read data from the stream to a buffer.
1234	<code>Seek</code>	Set file position.
1234	<code>Truncate</code>	Truncate the file on the current position.
1236	<code>Write</code>	Write data from a buffer to the stream.

67.12.3 TDosStream.Init

Synopsis: Instantiate a new instance of TDosStream.

Declaration: `constructor Init (FileName: FNameStr; Mode: Word)`

Visibility: default

Description: `Init` instantiates an instance of `TDosStream`. The name of the file that contains (or will contain) the data of the stream is given in `FileName`. The `Mode` parameter determines whether a new file should be created and what access rights you have on the file. It can be one of the following constants:

stCreate Creates a new file.

stOpenRead Read access only.

stOpenWrite Write access only.

stOpen Read and write access.

For an example, see `TDosStream.Truncate` ([1234](#)).

Errors: On error, `Status` (??) is set to `stInitError`, and `ErrorInfo` is set to the dos error code.

See also: `TDosStream.Done` ([1233](#))

67.12.4 TDosStream.Done

Synopsis: Closes the file and cleans up the instance.

Declaration: `destructor Done; Virtual`

Visibility: default

Description: `Done` closes the file if it was open and cleans up the instance of `TDosStream`.
for an example, see e.g. `TDosStream.Truncate` ([1234](#)).

Errors: None.

See also: `TDosStream.Init` ([1233](#)), `TDosStream.Close` ([1233](#))

67.12.5 TDosStream.Close

Synopsis: Close the file.

Declaration: `procedure Close; Virtual`

Visibility: default

Description: `Close` closes the file if it was open, and sets `Handle` to -1. Contrary to `Done` ([1233](#)) it does not clean up the instance of `TDosStream`

For an example, see `TDosStream.Open` ([1235](#)).

Errors: None.

See also: `TStream.Close` ([1263](#)), `TDosStream.Init` ([1233](#)), `TDosStream.Done` ([1233](#))

67.12.6 TDosStream.Truncate

Synopsis: Truncate the file on the current position.

Declaration: `procedure Truncate; Virtual`

Visibility: default

Description: If the status of the stream is `stOK`, then `Truncate` tries to truncate the stream size to the current file position.

Errors: If an error occurs, the stream's status is set to `stError` and `ErrorInfo` is set to the OS error code.

See also: `TStream.Truncate` ([1264](#)), `TStream.GetSize` ([1261](#))

Listing: `./examples/objectex/ex16.pp`

Program `ex16`;

{ Program to demonstrate the TStream.Truncate method }

Uses `Objects`;

Var `L : String`;

`P : PString`;

`S : PDosStream`; *{ Only one with Truncate implemented. }*

begin

`L := 'Some constant string';`

{ Buffer size of 100 }

`S := New(PDosStream, Init('test.dat', stcreate));`

Writeln ('Writing "', L, '" to stream with handle ', S^.Handle);

`S^.WriteStr(@L);`

`S^.WriteStr(@L);`

{ Close calls flush first }

`S^.Close;`

`S^.Open(stOpen);`

Writeln ('Size of stream is : ', S^.GetSize);

`P := S^.ReadStr;`

`L := P^;`

DisposeStr(P);

Writeln ('Read "', L, '" from stream with handle ', S^.Handle);

`S^.Truncate;`

Writeln ('Truncated stream. Size is : ', S^.GetSize);

`S^.Close;`

Dispose (S, Done);

end.

67.12.7 TDosStream.Seek

Synopsis: Set file position.

Declaration: `procedure Seek(Pos: LongInt); Virtual`

Visibility: default

Description: If the stream's status is `stOK`, then `Seek` sets the file position to `Pos`. `Pos` is a zero-based offset, counted from the beginning of the file.

Errors: In case an error occurs, the stream's status is set to `stSeekError`, and the OS error code is stored in `ErrorInfo`.

See also: `TStream.Seek` ([1265](#)), `TStream.GetPos` ([1261](#))

Listing: `./examples/objectex/ex17.pp`

Program `ex17`;

{ Program to demonstrate the TStream.Seek method }

Uses `Objects`;

Var `L : String`;

`Marker : Word`;

`P : PString`;

`S : PDosStream`;

begin

`L := 'Some constant string'`;

{ Buffer size of 100 }

`S := New(PDosStream, Init('test.dat', stcreate));`

Writeln ('Writing "', L, '" to stream.');

`S^.WriteStr(@L)`;

`Marker := S^.GetPos`;

Writeln ('Set marker at ', Marker);

`L := 'Some other constant String'`;

Writeln ('Writing "', L, '" to stream.');

`S^.WriteStr(@L)`;

`S^.Close`;

`S^.Open (stOpenRead)`;

Writeln ('Size of stream is : ', S^.GetSize);

Writeln ('Seeking to marker');

`S^.Seek(Marker)`;

`P := S^.ReadStr`;

`L := P^`;

DisposeStr(P);

Writeln ('Read "', L, '" from stream.');

`S^.Close`;

Dispose (S, Done);

end.

67.12.8 TDosStream.Open

Synopsis: Open the file stream.

Declaration: `procedure Open(OpenMode: Word); Virtual`

Visibility: default

Description: If the stream's status is `stOK`, and the stream is closed then `Open` re-opens the file stream with mode `OpenMode`. This call can be used after a `Close` ([1233](#)) call.

Errors: If an error occurs when re-opening the file, then `Status` is set to `stOpenError`, and the OS error code is stored in `ErrorInfo`

See also: `TStream.Open` ([1263](#)), `TDosStream.Close` ([1233](#))

Listing: ./examples/objectex/ex14.pp

Program ex14;

{ Program to demonstrate the TStream.Close method }

Uses Objects;

Var L : String;
 P : PString;
 S : PDosStream; *{ Only one with Close implemented. }*

begin
 L:='Some constant string';
 S:=New(PDosStream,Init('test.dat',stcreate));
 WriteLn ('Writing "',L,'" to stream with handle ',S^.Handle);
 S^.WriteStr(@L);
 S^.Close;
 WriteLn ('Closed stream. File handle is ',S^.Handle);
 S^.Open (stOpenRead);
 P:=S^.ReadStr;
 L:=P^;
 DisposeStr(P);
 WriteLn ('Read "',L,'" from stream with handle ',S^.Handle);
 S^.Close;
 Dispose (S,Done);
end.

67.12.9 TDosStream.Read

Synopsis: Read data from the stream to a buffer.

Declaration: `procedure Read(var Buf; Count: LongInt); Virtual`

Visibility: default

Description: If the Stream is open and the stream status is stOK then Read will read Count bytes from the stream and place them in Buf.

For an example, see TStream.Read ([1266](#)).

Errors: In case of an error, Status is set to StReadError, and ErrorInfo gets the OS specific error, or 0 when an attempt was made to read beyond the end of the stream.

See also: TStream.Read ([1266](#)), TDosStream.Write ([1236](#))

67.12.10 TDosStream.Write

Synopsis: Write data from a buffer to the stream.

Declaration: `procedure Write(var Buf; Count: LongInt); Virtual`

Visibility: default

Description: If the Stream is open and the stream status is stOK then Write will write Count bytes from Buf and place them in the stream.

For an example, see TStream.Read ([1266](#)).

Errors: In case of an error, `Status` is set to `StWriteError`, and `ErrorInfo` gets the OS specific error.

See also: `TStream.Write` ([1266](#)), `TDosStream.Read` ([1236](#))

67.13 TMemoryStream

67.13.1 Description

The `TMemoryStream` object implements a stream that stores it's data in memory. The data is stored on the heap, with the possibility to specify the maximum amount of data, and the size of the memory blocks being used.

See also: `TStream` ([1259](#))

67.13.2 Method overview

Page	Method	Description
1237	<code>Done</code>	Clean up memory and destroy the object instance.
1237	<code>Init</code>	Initialize memory stream, reserves memory for stream data.
1238	<code>Read</code>	Read data from the stream to a location in memory.
1238	<code>Truncate</code>	Set the stream size to the current position.
1239	<code>Write</code>	Write data to the stream.

67.13.3 TMemoryStream.Init

Synopsis: Initialize memory stream, reserves memory for stream data.

Declaration: constructor `Init(ALimit: LongInt; ABlockSize: Word)`

Visibility: default

Description: `Init` instantiates a new `TMemoryStream` object. The `memorystreamobject` will initially allocate at least `ALimit` bytes memory, divided into memory blocks of size `ABlockSize`. The number of blocks needed to get to `ALimit` bytes is rounded up.

By default, the number of blocks is 1, and the size of a block is 8192. This is selected if you specify 0 as the blocksize.

For an example, see e.g `TStream.CopyFrom` ([1267](#)).

Errors: If the stream cannot allocate the initial memory needed for the memory blocks, then the stream's status is set to `stInitError`.

See also: `TMemoryStream.Done` ([1237](#))

67.13.4 TMemoryStream.Done

Synopsis: Clean up memory and destroy the object instance.

Declaration: destructor `Done; Virtual`

Visibility: default

Description: `Done` releases the memory blocks used by the stream, and then cleans up the memory used by the stream object itself.

For an example, see e.g `TStream.CopyFrom` ([1267](#)).

Errors: None.

See also: TMemoryStream.Init ([1237](#))

67.13.5 TMemoryStream.Truncate

Synopsis: Set the stream size to the current position.

Declaration: `procedure Truncate; Virtual`

Visibility: default

Description: `Truncate` sets the size of the memory stream equal to the current position. It de-allocates any memory-blocks that are no longer needed, so that the new size of the stream is the current position in the stream, rounded up to the first multiple of the stream blocksize.

Errors: If an error occurs during memory de-allocation, the stream's status is set to `stError`

See also: TStream.Truncate ([1264](#))

Listing: `./examples/objectex/ex20.pp`

Program `ex20`;

{ Program to demonstrate the TMemoryStream.Truncate method }

Uses Objects;

Var `L : String`;

`P : PString`;

`S : PMemoryStream`;

`I : Longint`;

begin

`L := 'Some constant string';`

{ Buffer size of 100 }

`S := New(PMemoryStream, Init(1000, 100));`

`Writeln ('Writing 100 times "", L, "" to stream.');`

For `I := 1 to 100 do`

`S^.WriteStr(@L);`

`Writeln ('Finished.');`

`S^.Seek(100);`

`S^.Truncate;`

`Writeln ('Truncated at byte 100.');`

`Dispose (S, Done);`

`Writeln ('Finished.');`

end.

67.13.6 TMemoryStream.Read

Synopsis: Read data from the stream to a location in memory.

Declaration: `procedure Read(var Buf; Count: LongInt); Virtual`

Visibility: default

Description: `Read` reads `Count` bytes from the stream to `Buf`. It updates the position of the stream.

For an example, see TStream.Read ([1266](#)).

Errors: If there is not enough data available, no data is read, and the stream's status is set to `stReadError`.

See also: `TStream.Read` ([1266](#)), `TMemoryStream.Write` ([1239](#))

67.13.7 TMemoryStream.Write

Synopsis: Write data to the stream.

Declaration: `procedure Write(var Buf; Count: LongInt); Virtual`

Visibility: default

Description: Write copies `Count` bytes from `Buf` to the stream. It updates the position of the stream.

If not enough memory is available to hold the extra `Count` bytes, then the stream will try to expand, by allocating as much blocks with size `BlkSize` (as specified in the constructor call `Init` ([1237](#))) as needed.

For an example, see `TStream.Read` ([1266](#)).

Errors: If the stream cannot allocate more memory, then the status is set to `stWriteError`

See also: `TStream.Write` ([1266](#)), `TMemoryStream.Read` ([1238](#))

67.14 TObject

67.14.1 Description

This type serves as the basic object for all other objects in the `Objects` unit.

67.14.2 Method overview

Page	Method	Description
1240	<code>Done</code>	Destroy an object.
1240	<code>Free</code>	Destroy an object and release all memory.
1239	<code>Init</code>	Construct (initialize) a new object.
1240	<code>Is_Object</code>	Check whether a pointer points to an object.

67.14.3 TObject.Init

Synopsis: Construct (initialize) a new object.

Declaration: `constructor Init`

Visibility: default

Description: Instantiates a new object of type `TObject`. It fills the instance up with `Zero` bytes.

For an example, see `Free` ([1240](#))

Errors: None.

See also: `TObject.Free` ([1240](#)), `TObject.Done` ([1240](#))

67.14.4 TObject.Free

Synopsis: Destroy an object and release all memory.

Declaration: `procedure Free`

Visibility: `default`

Description: `Free` calls the destructor of the object, and releases the memory occupied by the instance of the object.

Errors: No checking is performed to see whether `self` is `nil` and whether the object is indeed allocated on the heap.

See also: `TObject.Init` ([1239](#)), `TObject.Done` ([1240](#))

Listing: `./examples/objectex/ex7.pp`

```

program ex7;

{ Program to demonstrate the TObject.Free call }

Uses Objects;

Var O : PObject;

begin
  // Allocate memory for object.
  O := New(PObject, Init);
  // Free memory of object.
  O^.free;
end.

```

67.14.5 TObject.Is_Object

Synopsis: Check whether a pointer points to an object.

Declaration: `function Is_Object (P: Pointer) : Boolean`

Visibility: `default`

Description: `Is_Object` returns `True` if the pointer `P` points to an instance of a `TObject` descendent, it returns `false` otherwise.

67.14.6 TObject.Done

Synopsis: Destroy an object.

Declaration: `destructor Done; Virtual`

Visibility: `default`

Description: `Done`, the destructor of `TObject` does nothing. It is mainly intended to be used in the `TObject.Free` ([1240](#)) method.

The destructor `Done` does not free the memory occupied by the object.

Errors: None.

See also: `TObject.Free` ([1240](#)), `TObject.Init` ([1239](#))

Listing: `./examples/objectex/ex8.pp`

```

program ex8;

{ Program to demonstrate the TObject.Done call }

Uses Objects;

Var O : PObject;

begin
  // Allocate memory for object.
  O:=New(PObject,Init);
  O^.Done;
end.

```

67.15 TPoint

67.15.1 Description

Record describing a point in a 2 dimensional plane.

67.16 TRect

67.16.1 Description

Describes a rectangular region in a plane.

67.16.2 Method overview

Page	Method	Description
1246	Assign	Set rectangle corners.
1243	Contains	Determine if a point is inside the rectangle.
1243	Copy	Copy cornerpoints from another rectangle.
1241	Empty	Is the surface of the rectangle zero.
1242	Equals	Do the corners of the rectangles match.
1246	Grow	Expand rectangle with certain size.
1244	Intersect	Reduce rectangle to intersection with another rectangle.
1245	Move	Move rectangle along a vector.
1244	Union	Enlarges rectangle to encompass another rectangle.

67.16.3 TRect.Empty

Synopsis: Is the surface of the rectangle zero.

Declaration: `function Empty : Boolean`

Visibility: default

Description: `Empty` returns `True` if the rectangle defined by the corner points A, B has zero or negative surface.

Errors: None.

See also: `TRect.Equals` ([1242](#)), `TRect.Contains` ([1243](#))

Listing: `./examples/objectex/ex1.pp`

Program `ex1`;

{ Program to demonstrate TRect.Empty }

Uses `objects`;

Var `ARect,BRect` : `TRect`;
 `P` : `TPoint`;

```
begin
  With ARect.A do
    begin
      X:=10;
      Y:=10;
    end;
  With ARect.B do
    begin
      X:=20;
      Y:=20;
    end;
  { Offset B by (5,5) }
  With BRect.A do
    begin
      X:=15;
      Y:=15;
    end;
  With BRect.B do
    begin
      X:=25;
      Y:=25;
    end;
  { Point }
  With P do
    begin
      X:=15;
      Y:=15;
    end;
  Writeln ('A empty : ',ARect.Empty);
  Writeln ('B empty : ',BRect.Empty);
  Writeln ('A Equals B : ',ARect.Equals(BRect));
  Writeln ('A Contains (15,15) : ',ARect.Contains(P));
end.
```

67.16.4 TRect.Equals

Synopsis: Do the corners of the rectangles match.

Declaration: `function Equals(R: TRect) : Boolean`

Visibility: `default`

Description: `Equals` returns `True` if the rectangle has the same corner points `A, B` as the rectangle `R`, and `False` otherwise.

For an example, see `TRect.Empty` (1241)

Errors: None.

See also: `TRect.Empty` (1241), `TRect.Contains` (1243)

67.16.5 TRect.Contains

Synopsis: Determine if a point is inside the rectangle.

Declaration: `function Contains(P: TPoint) : Boolean`

Visibility: default

Description: `Contains` returns `True` if the point `P` is contained in the rectangle (including borders), `False` otherwise.

Errors: None.

See also: `TRect.Intersect` (1244), `TRect.Equals` (1242)

67.16.6 TRect.Copy

Synopsis: Copy cornerpoints from another rectangle.

Declaration: `procedure Copy(R: TRect)`

Visibility: default

Description: Assigns the rectangle `R` to the object. After the call to `Copy`, the rectangle `R` has been copied to the object that invoked `Copy`.

Errors: None.

See also: `TRect.Assign` (1246)

Listing: `./examples/objectex/ex2.pp`

Program `ex2`;

{ Program to demonstrate TRect.Copy }

Uses `objects`;

Var `ARect, BRect, CRect` : `TRect`;

begin

`ARect.Assign(10,10,20,20);`

`BRect.Assign(15,15,25,25);`

`CRect.Copy(ARect);`

If `ARect.Equals(CRect)` **Then**

Writeln ('ARect equals CRect')

Else

Writeln ('ARect does not equal CRect !');

end.

67.16.7 TRect.Union

Synopsis: Enlarges rectangle to encompass another rectangle.

Declaration: `procedure Union (R: TRect)`

Visibility: default

Description: `Union` enlarges the current rectangle so that it becomes the union of the current rectangle with the rectangle `R`.

Errors: None.

See also: `TRect.Intersect` ([1244](#))

Listing: `./examples/objectex/ex3.pp`

Program `ex3`;

{ Program to demonstrate TRect.Union }

Uses `objects`;

Var `ARect,BRect,CRect` : `TRect`;

begin

`ARect.Assign(10,10,20,20);`

`BRect.Assign(15,15,25,25);`

{ CRect is union of ARect and BRect }

`CRect.Assign(10,10,25,25);`

{ Calculate it explicitly }

`ARect.Union(BRect);`

If `ARect.Equals(CRect)` **Then**

`Writeln ('ARect equals CRect')`

Else

`Writeln ('ARect does not equal CRect !');`

end.

67.16.8 TRect.Intersect

Synopsis: Reduce rectangle to intersection with another rectangle.

Declaration: `procedure Intersect (R: TRect)`

Visibility: default

Description: `Intersect` makes the intersection of the current rectangle with `R`. If the intersection is empty, then the rectangle is set to the empty rectangle at coordinate (0,0).

Errors: None.

See also: `TRect.Union` ([1244](#))

Listing: `./examples/objectex/ex4.pp`

Program ex4;

{ Program to demonstrate TRect.Intersect }

Uses objects;

Var ARect,BRect,CRect : TRect;

begin

```
ARect.Assign(10,10,20,20);
BRect.Assign(15,15,25,25);
{ CRect is intersection of ARect and BRect }
CRect.Assign(15,15,20,20);
{ Calculate it explicitly }
ARect.Intersect(BRect);
If ARect.Equals(CRect) Then
  Writeln ('ARect equals CRect')
Else
  Writeln ('ARect does not equal CRect !');
BRect.Assign(25,25,30,30);
ARect.Intersect(BRect);
If ARect.Empty Then
  Writeln ('ARect is empty');
```

end.

67.16.9 TRect.Move

Synopsis: Move rectangle along a vector.

Declaration: `procedure Move (ADX: Sw_Integer; ADY: Sw_Integer)`

Visibility: default

Description: `Move` moves the current rectangle along a vector with components (ADX, ADY) . It adds ADX to the X-coordinate of both corner points, and ADY to both end points.

Errors: None.

See also: TRect.Grow ([1246](#))

Listing: ./examples/objectex/ex5.pp

Program ex5;

{ Program to demonstrate TRect.Move }

Uses objects;

Var ARect,BRect : TRect;

begin

```
ARect.Assign(10,10,20,20);
ARect.Move(5,5);
// Brect should be where new ARect is.
BRect.Assign(15,15,25,25);
```

```

If ARect.Equals(BRect) Then
  Writeln ('ARect equals BRect')
Else
  Writeln ('ARect does not equal BRect !');
end.

```

67.16.10 TRect.Grow

Synopsis: Expand rectangle with certain size.

Declaration: `procedure Grow(ADX: Sw_Integer; ADY: Sw_Integer)`

Visibility: default

Description: `Grow` expands the rectangle with an amount `ADX` in the X direction (both on the left and right side of the rectangle, thus adding a length $2*ADX$ to the width of the rectangle), and an amount `ADY` in the Y direction (both on the top and the bottom side of the rectangle, adding a length $2*ADY$ to the height of the rectangle).

`ADX` and `ADY` can be negative. If the resulting rectangle is empty, it is set to the empty rectangle at `(0, 0)`.

Errors: None.

See also: `TRect.Move` ([1245](#))

Listing: `./examples/objectex/ex6.pp`

Program ex6;

{ Program to demonstrate TRect.Grow }

Uses objects;

Var ARect,BRect : TRect;

begin

ARect.Assign(10,10,20,20);

ARect.Grow(5,5);

// Brect should be where new ARect is.

BRect.Assign(5,5,25,25);

If ARect.Equals(BRect) **Then**

Writeln ('ARect equals BRect')

Else

Writeln ('ARect does not equal BRect !');

end.

67.16.11 TRect.Assign

Synopsis: Set rectangle corners.

Declaration: `procedure Assign(XA: Sw_Integer; YA: Sw_Integer; XB: Sw_Integer; YB: Sw_Integer)`

Visibility: default

Description: Assign sets the corner points of the rectangle to (XA, YA) and (Xb, Yb).

For an example, see TRect.Copy ([1243](#)).

Errors: None.

See also: TRect.Copy ([1243](#))

67.17 TResourceCollection

67.17.1 Description

A TResourceCollection manages a collection of resource names. It stores the position and the size of a resource, as well as the name of the resource. It stores these items in records that look like this:

```
TYPE
  TResourceItem = packed RECORD
    Posn: LongInt;
    Size: LongInt;
    Key : String;
  End;
  PResourceItem = ^TResourceItem;
```

It overrides some methods of TStringCollection in order to accomplish this.

Remark Remark that the TResourceCollection manages the names of the resources and their associated positions and sizes, it doesn't manage the resources themselves.

67.17.2 Method overview

Page	Method	Description
1248	FreeItem	Release memory occupied by item.
1248	GetItem	Read an item from the stream.
1247	KeyOf	Return the key of an item in the collection.
1248	PutItem	Write an item to the stream.

67.17.3 TResourceCollection.KeyOf

Synopsis: Return the key of an item in the collection.

Declaration: function KeyOf(Item: Pointer) : Pointer; Virtual

Visibility: default

Description: KeyOf returns the key of an item in the collection. For resources, the key is a pointer to the string with the resource name.

Errors: None.

See also: TStringCollection.Compare ([1268](#))

67.17.4 TResourceCollection.GetItem

Synopsis: Read an item from the stream.

Declaration: `function GetItem(var S: TStream) : Pointer; Virtual`

Visibility: default

Description: `GetItem` reads a resource item from the stream `S`. It reads the position, size and name from the stream, in that order. It DOES NOT read the resource itself from the stream.

The resulting item is not inserted in the collection. This call is mainly for internal use by the `TCollection.Load (1218)` method.

Errors: Errors returned are those by `TStream.Read (1266)`

See also: `TCollection.Load (1218)`, `TStream.Read (1266)`

67.17.5 TResourceCollection.FreeItem

Synopsis: Release memory occupied by item.

Declaration: `procedure FreeItem(Item: Pointer); Virtual`

Visibility: default

Description: `FreeItem` releases the memory occupied by `Item`. It de-allocates the name, and then the resource item record.

It does NOT remove the item from the collection.

Errors: None.

See also: `TCollection.FreeItem (1228)`

67.17.6 TResourceCollection.PutItem

Synopsis: Write an item to the stream.

Declaration: `procedure PutItem(var S: TStream; Item: Pointer); Virtual`

Visibility: default

Description: `PutItem` writes `Item` to the stream `S`. It does this by writing the position and size and name of the resource item to the stream.

This method is used primarily by the `Store (1232)` method.

Errors: Errors returned are those by `TStream.Write (1266)`.

See also: `TCollection.Store (1232)`

67.18 TResourceFile

67.18.1 Description

`TResourceFile (1248)` represents the resources in a binary file image.

67.18.2 Method overview

Page	Method	Description
1249	Count	Number of resources in the file.
1251	Delete	Delete a resource from the file.
1249	Done	Destroy the instance and remove it from memory.
1250	Flush	Writes the resources to the stream.
1250	Get	Return a resource by key name.
1249	Init	Instantiate a new instance.
1250	KeyAt	Return the key of the item at a certain position.
1251	Put	Set a resource by key name.
1250	SwitchTo	Write resources to a new stream.

67.18.3 TResourceFile.Init

Synopsis: Instantiate a new instance.

Declaration: `constructor Init (AStream: PStream)`

Visibility: default

Description: `Init` instantiates a new instance of a `TResourceFile` object. If `AStream` is not nil then it is considered as a stream describing an executable image on disk.

`Init` will try to position the stream on the start of the resources section, and read all resources from the stream.

Errors: None.

See also: `TResourceFile.Done` ([1249](#))

67.18.4 TResourceFile.Done

Synopsis: Destroy the instance and remove it from memory.

Declaration: `destructor Done; Virtual`

Visibility: default

Description: `Done` cleans up the instance of the `TResourceFile` Object. If `Stream` was specified at initialization, then `Stream` is disposed of too.

Errors: None.

See also: `TResourceFile.Init` ([1249](#))

67.18.5 TResourceFile.Count

Synopsis: Number of resources in the file.

Declaration: `function Count : Sw_Integer`

Visibility: default

Description: `Count` returns the number of resources. If no resources were read, zero is returned.

Errors: None.

See also: `TResourceFile.Init` ([1249](#))

67.18.6 TResourceFile.KeyAt

Synopsis: Return the key of the item at a certain position.

Declaration: `function KeyAt (I: Sw_Integer) : string`

Visibility: default

Description: `KeyAt` returns the key (the name) of the `I`-th resource.

Errors: In case `I` is invalid, `TCollection.Error` will be executed.

See also: `TResourceFile.Get` ([1250](#))

67.18.7 TResourceFile.Get

Synopsis: Return a resource by key name.

Declaration: `function Get (Key: string) : PObject`

Visibility: default

Description: `Get` returns a pointer to a instance of a resource identified by `Key`. If `Key` cannot be found in the list of resources, then `Nil` is returned.

Errors: Errors returned may be those by `TStream.Get`

67.18.8 TResourceFile.SwitchTo

Synopsis: Write resources to a new stream.

Declaration: `function SwitchTo (AStream: PStream; Pack: Boolean) : PStream`

Visibility: default

Description: `SwitchTo` switches to a new stream to hold the resources in. `AStream` will be the new stream after the call to `SwitchTo`.

If `Pack` is true, then all the known resources will be copied from the current stream to the new stream (`AStream`). If `Pack` is False, then only the current resource is copied.

The return value is the value of the original stream: `Stream`.

The `Modified` flag is set as a consequence of this call.

Errors: Errors returned can be those of `TStream.Read` ([1266](#)) and `TStream.Write` ([1266](#)).

See also: `TResourceFile.Flush` ([1250](#))

67.18.9 TResourceFile.Flush

Synopsis: Writes the resources to the stream.

Declaration: `procedure Flush`

Visibility: default

Description: If the `Modified` flag is set to `True`, then `Flush` writes the resources to the stream `Stream`. It sets the `Modified` flag to true after that.

Errors: Errors can be those by `TStream.Seek` ([1265](#)) and `TStream.Write` ([1266](#)).

See also: `TResourceFile.SwitchTo` ([1250](#))

67.18.10 TResourceFile.Delete

Synopsis: Delete a resource from the file.

Declaration: `procedure Delete(Key: string)`

Visibility: `default`

Description: `Delete` deletes the resource identified by `Key` from the collection. It sets the `Modified` flag to `true`.

Errors: `None`.

See also: `TResourceFile.Flush` ([1250](#))

67.18.11 TResourceFile.Put

Synopsis: Set a resource by key name.

Declaration: `procedure Put(Item: PObject; Key: string)`

Visibility: `default`

Description: `Put` sets the resource identified by `Key` to `Item`. If no such resource exists, a new one is created. The item is written to the stream.

Errors: Errors returned may be those by `TStream.Put` ([1264](#)) and `TStream.Seek`

See also: `Get` ([1250](#))

67.19 TSortedCollection

67.19.1 Description

`TSortedCollection` is an abstract class, implementing a sorted collection. You should never use an instance of `TSortedCollection` directly, instead you should declare a descendent type, and override the `Compare` ([1253](#)) method.

Because the collection is ordered, `TSortedCollection` overrides some `TCollection` methods, to provide faster routines for lookup.

The `Compare` ([1253](#)) method decides how elements in the collection should be ordered. Since `TCollection` has no way of knowing how to order pointers, you must override the compare method.

Additionally, `TCollection` provides a means to filter out duplicates. if you set `Duplicates` to `False` (the default) then duplicates will not be allowed.

The example below defines a descendent of `TSortedCollection` which is used in the examples.

67.19.2 Method overview

Page	Method	Description
1253	Compare	Compare two items in the collection.
1253	IndexOf	Return index of an item in the collection.
1252	Init	Instantiates a new instance of a <code>TSortedCollection</code> .
1255	Insert	Insert new item in collection.
1252	KeyOf	Return the key of an item.
1252	Load	Instantiates a new instance of a <code>TSortedCollection</code> and loads it from stream.
1254	Search	Search for item with given key.
1256	Store	Write the collection to the stream.

67.19.3 TSortedCollection.Init

Synopsis: Instantiates a new instance of a `TSortedCollection`.

Declaration: `constructor Init (ALimit: Sw_Integer; ADelta: Sw_Integer)`

Visibility: default

Description: `Init` calls the inherited constructor (see `TCollection.Init` ([1218](#))) and sets the `Duplicates` flag to false.

You should not call this method directly, since `TSortedCollection` is a abstract class. Instead, the descendent classes should call it via the `inherited` keyword.

Errors: None.

See also: `TSortedCollection.Load` ([1252](#)), `TCollection.Done` ([1219](#))

67.19.4 TSortedCollection.Load

Synopsis: Instantiates a new instance of a `TSortedCollection` and loads it from stream.

Declaration: `constructor Load (var S: TStream)`

Visibility: default

Description: `Load` calls the inherited constructor (see `TCollection.Load` ([1218](#))) and reads the `Duplicates` flag from the stream..

You should not call this method directly, since `TSortedCollection` is a abstract class. Instead, the descendent classes should call it via the `inherited` keyword.

For an example, see `TCollection.Load` ([1218](#)).

Errors: None.

See also: `TSortedCollection.Init` ([1252](#)), `TCollection.Done` ([1219](#))

67.19.5 TSortedCollection.KeyOf

Synopsis: Return the key of an item.

Declaration: `function KeyOf (Item: Pointer) : Pointer; Virtual`

Visibility: default

Description: `KeyOf` returns the key associated with `Item`. `TSortedCollection` returns the item itself as the key, descendent objects can override this method to calculate a (unique) key based on the item passed (such as hash values).

`Keys` are used to sort the objects, they are used to search and sort the items in the collection. If descendent types override this method then it allows possibly for faster search/sort methods based on keys rather than on the objects themselves.

Errors: None.

See also: `TSortedCollection.IndexOf` ([1253](#)), `TSortedCollection.Compare` ([1253](#))

67.19.6 TSortedCollection.IndexOf

Synopsis: Return index of an item in the collection.

Declaration: `function IndexOf(Item: Pointer) : Sw_Integer; Virtual`

Visibility: default

Description: `IndexOf` returns the index of `Item` in the collection. It searches for the object based on it's key. If duplicates are allowed, then it returns the index of last object that matches `Item`.

In case `Item` is not found in the collection, -1 is returned.

For an example, see `TCollection.IndexOf` ([1220](#))

Errors: None.

See also: `TSortedCollection.Search` ([1254](#)), `TSortedCollection.Compare` ([1253](#))

67.19.7 TSortedCollection.Compare

Synopsis: Compare two items in the collection.

Declaration: `function Compare(Key1: Pointer; Key2: Pointer) : Sw_Integer; Virtual`

Visibility: default

Description: `Compare` is an abstract method that should be overridden by descendent objects in order to compare two items in the collection. This method is used in the `Search` ([1254](#)) method and in the `Insert` ([1255](#)) method to determine the ordering of the objects.

The function should compare the two keys of items and return the following function results:

Result < 0 If `Key1` is logically before `Key2` (`Key1 < Key2`)

Result = 0 If `Key1` and `Key2` are equal. (`Key1 = Key2`)

Result > 0 If `Key1` is logically after `Key2` (`Key1 > Key2`)

Errors: An 'abstract run-time error' will be generated if you call `TSortedCollection.Compare` directly.

See also: `TSortedCollection.IndexOf` ([1253](#)), `TSortedCollection.Search` ([1254](#))

Listing: `./examples/objectex/mysortc.pp`

Unit MySortC;

Interface

Uses Objects;

Type

PMySortedCollection = ^TMySortedCollection;
 TMySortedCollection = **Object**(TSortedCollection)
 Function Compare (Key1,Key2 : Pointer): Sw_integer; **virtual**;
 end;

Implementation

Uses MyObject;

Function TMySortedCollection.Compare (Key1,Key2 : Pointer) :sw_integer;

begin

 Compare:=PMyobject(Key1)^.GetField – PMyObject(Key2)^.GetField;
end;

end.

67.19.8 TSortedCollection.Search

Synopsis: Search for item with given key.

Declaration: `function Search(Key: Pointer; var Index: Sw_Integer) : Boolean; Virtual`

Visibility: default

Description: `Search` looks for the item with key `Key` and returns the position of the item (if present) in the collection in `Index`.

Instead of a linear search as `TCollection` does, `TSortedCollection` uses a binary search based on the keys of the objects. It uses the `Compare` ([1253](#)) function to implement this search.

If the item is found, `Search` returns `True`, otherwise `False` is returned.

Errors: None.

See also: `TCollection.IndexOf` ([1220](#))

Listing: ./examples/objectex/ex36.pp

Program ex36;

{ Program to demonstrate the TSortedCollection.Insert method }

Uses Objects,MyObject,MySortC;

{ For TMyObject,TMySortedCollection definition and registration }

Var C : PSortedCollection;

 M : PMyObject;

 I : Longint;

Procedure PrintField (Dummy: Pointer;P : PMyObject);

```

begin
  Writeln ('Field : ',P^.GetField);
end;

begin
  Randomize;
  C:=New(PMySortedCollection,Init(120,10));
  C^.Duplicates:=True;
  Writeln ('Inserting 100 records at random places. ');
  For I:=1 to 100 do
    begin
      M:=New(PMyObject,Init);
      M^.SetField(Random(100));
      C^.Insert(M)
    end;
  M:=New(PMyObject,Init);
  Repeat;
    Write ('Value to search for (-1 stops) : ');
    read (I);
    If I<>-1 then
      begin
        M^.SetField(i);
        If Not C^.Search (M,I) then
          Writeln ('No such value found')
        else
          begin
            Write ('Value ',PMyObject(C^.At(I))^GetField);
            Writeln (' present at position ',I);
          end;
        end;
      Until I=-1;
    Dispose(M,Done);
    Dispose(C,Done);
  end.

```

67.19.9 TSortedCollection.Insert

Synopsis: Insert new item in collection.

Declaration: `procedure Insert (Item: Pointer); Virtual`

Visibility: default

Description: `Insert` inserts an item in the collection at the correct position, such that the collection is ordered at all times. You should never use `Atinsert` ([1231](#)), since then the collection ordering is not guaranteed.

If `Item` is already present in the collection, and `Duplicates` is `False`, the item will not be inserted.

Errors: None.

See also: `TCollection.AtInsert` ([1231](#))

Listing: `./examples/objectex/ex35.pp`

Program ex35;

{ Program to demonstrate the TSortedCollection.Insert method }

Uses Objects, MyObject, MySortC;

{ For TMyObject, TMySortedCollection definition and registration }

Var C : PSortedCollection;

 M : PMyObject;

 I : Longint;

Procedure PrintField (Dummy: Pointer; P : PMyObject);

begin

Writeln ('Field : ', P^.GetField);

end;

begin

Randomize;

 C := **New**(PMySortedCollection, Init(120, 10));

Writeln ('Inserting 100 records at random places.');

For I := 1 **to** 100 **do**

begin

 M := **New**(PMyObject, Init);

 M^.SetField(**Random**(100));

 C^.Insert(M)

end;

Writeln ('Values : ');

 C^.Foreach(@PrintField);

Dispose(C, Done);

end.

67.19.10 TSortedCollection.Store

Synopsis: Write the collection to the stream.

Declaration: `procedure Store (var S : TStream)`

Visibility: default

Description: `Store` writes the collection to the stream `S`. It does this by calling the inherited `TCollection.Store` ([1232](#)), and then writing the `Duplicates` flag to the stream.

After a `Store`, the collection can be loaded from the stream with the constructor `Load` ([1252](#))

For an example, see `TCollection.Load` ([1218](#)).

Errors: Errors can be those of `TStream.Put` ([1264](#)).

See also: `TSortedCollection.Load` ([1252](#))

67.20 TStrCollection

67.20.1 Description

The `TStrCollection` object manages a sorted collection of null-terminated strings (pchar strings). To this end, it overrides the `Compare` ([1253](#)) method of `TSortedCollection`, and it introduces methods to read/write strings from a stream.

67.20.2 Method overview

Page	Method	Description
1257	<code>Compare</code>	Compare two strings in the collection.
1258	<code>FreeItem</code>	Free null-terminated string from the collection.
1258	<code>GetItem</code>	Read a null-terminated string from the stream.
1258	<code>PutItem</code>	Write a null-terminated string to the stream.

67.20.3 TStrCollection.Compare

Synopsis: Compare two strings in the collection.

Declaration: `function Compare(Key1: Pointer; Key2: Pointer) : Sw_Integer; Virtual`

Visibility: default

Description: `TStrCollection` overrides the `Compare` function so it compares the two keys as if they were pointers to strings. The compare is done case sensitive. It returns

-1 if the first string is alphabetically earlier than the second string.

0 if the two strings are equal.

1 if the first string is alphabetically later than the second string.

Errors: None.

See also: `TSortedCollection.Compare` ([1253](#))

Listing: `./examples/objectex/ex38.pp`

Program `ex38`;

{ Program to demonstrate the TStrCollection.Compare method }

Uses `Objects, Strings`;

Var `C : PStrCollection`;

`S : String`;

`I : longint`;

`P : Pchar`;

begin

`Randomize`;

`C := New(PStrCollection, Init(120, 10))`;

`C^.Duplicates := True`; *{ Duplicates allowed }*

Writeln ('Inserting 100 records at random places.');

For `I := 1 to 100 do`

`begin`

`Str(Random(100), S)`;

```

S:='String with value '+S;
P:=StrAlloc(Length(S)+1);
C^.Insert(StrPCopy(P,S));
end;
For I:=0 to 98 do
  With C^ do
    If Compare (At(I),At(I+1))=0 then
      Writeln ('Duplicate string found at position ',I);
    Dispose(C,Done);
  end.

```

67.20.4 TStrCollection.GetItem

Synopsis: Read a null-terminated string from the stream.

Declaration: `function GetItem(var S: TStream) : Pointer; Virtual`

Visibility: default

Description: `GetItem` reads a null-terminated string from the stream `S` and returns a pointer to it. It doesn't insert the string in the collection.

This method is primarily introduced to be able to load and store the collection from and to a stream.

Errors: The errors returned are those of `TStream.StrRead` ([1260](#)).

See also: `TStrCollection.PutItem` ([1258](#))

67.20.5 TStrCollection.FreeItem

Synopsis: Free null-terminated string from the collection.

Declaration: `procedure FreeItem(Item: Pointer); Virtual`

Visibility: default

Description: `TStrCollection` overrides `FreeItem` so that the string pointed to by `Item` is disposed from memory.

Errors: None.

See also: `TCollection.FreeItem` ([1228](#))

67.20.6 TStrCollection.PutItem

Synopsis: Write a null-terminated string to the stream.

Declaration: `procedure PutItem(var S: TStream; Item: Pointer); Virtual`

Visibility: default

Description: `PutItem` writes the string pointed to by `Item` to the stream `S`.

This method is primarily used in the `Load` and `Store` methods, and should not be used directly.

Errors: Errors are those of `TStream.StrWrite` ([1265](#)).

See also: `TStrCollection.GetItem` ([1258](#))

67.21 TStream

67.21.1 Description

The `TStream` object is the ancestor for all streaming objects, i.e. objects that have the capability to store and retrieve data.

It defines a number of methods that are common to all objects that implement streaming, many of them are virtual, and are only implemented in the descendent types.

Programs should not instantiate objects of type `TStream` directly, but instead instantiate a descendant type, such as `TDosStream`, `TMemoryStream`.

See also: `PStream` ([1204](#)), `TDosStream` ([1232](#)), `TMemoryStream` ([1237](#))

67.21.2 Method overview

Page	Method	Description
1263	<code>Close</code>	Close the stream.
1267	<code>CopyFrom</code>	Copy data from another stream.
1265	<code>Error</code>	Set stream status.
1264	<code>Flush</code>	Flush the stream data from the buffer, if any.
1259	<code>Get</code>	Read an object definition from the stream.
1261	<code>GetPos</code>	Return current position in the stream.
1261	<code>GetSize</code>	Return the size of the stream.
1259	<code>Init</code>	Constructor for <code>TStream</code> instance.
1263	<code>Open</code>	Open the stream.
1264	<code>Put</code>	Write an object to the stream.
1266	<code>Read</code>	Read data from stream to buffer.
1262	<code>ReadStr</code>	Read a shortstring from the stream.
1263	<code>Reset</code>	Reset the stream.
1265	<code>Seek</code>	Set stream position.
1260	<code>StrRead</code>	Read a null-terminated string from the stream.
1265	<code>StrWrite</code>	Write a null-terminated string to the stream.
1264	<code>Truncate</code>	Truncate the stream size on current position.
1266	<code>Write</code>	Write a number of bytes to the stream.
1265	<code>WriteStr</code>	Write a pascal string to the stream.

67.21.3 TStream.Init

Synopsis: Constructor for `TStream` instance.

Declaration: `constructor Init`

Visibility: `default`

Description: `Init` initializes a `TStream` instance. Descendent streams should always call the inherited `Init`.

67.21.4 TStream.Get

Synopsis: Read an object definition from the stream.

Declaration: `function Get : PObject`

Visibility: `default`

Description: `Get` reads an object definition from a stream, and returns a pointer to an instance of this object.

Errors: On error, `TStream.Status` (??) is set, and `NIL` is returned.

See also: `TStream.Put` ([1264](#))

Listing: `./examples/objectex/ex9.pp`

Program `ex9`;

{ Program to demonstrate TStream.Get and TStream.Put }

Uses `Objects, MyObject`; *{ Definition and registration of TMyObject }*

Var `Obj` : `PMMyObject`;
 `S` : `PStream`;

begin

```

Obj:=New(PMyObject,Init);
Obj^.SetField($1111) ;
Writeln ('Field value : ',Obj^.GetField);
{ Since Stream is an abstract type, we instantiate a TMemoryStream }
S:=New(PMemoryStream,Init(100,10));
S^.Put(Obj);
Writeln ('Disposing object');
S^.Seek(0);
Dispose(Obj,Done);
Writeln ('Reading object');
Obj:=PMMyObject(S^.Get);
Writeln ('Field Value : ',Obj^.GetField);
Dispose(Obj,Done);

```

end.

67.21.5 TStream.StrRead

Synopsis: Read a null-terminated string from the stream.

Declaration: `function StrRead` : `PChar`

Visibility: `default`

Description: `StrRead` reads a string from the stream, allocates memory for it, and returns a pointer to a null-terminated copy of the string on the heap.

Errors: On error, `Nil` is returned.

See also: `TStream.StrWrite` ([1265](#)), `TStream.ReadStr` ([1262](#))

Listing: `./examples/objectex/ex10.pp`

Program `ex10`;

```

{
Program to demonstrate the TStream.StrRead TStream.StrWrite functions
}

```

Uses `objects`;

```

Var P : PChar;
      S : PStream;

begin
  P:='Constant Pchar string';
  Writeln ('Writing to stream : "',P,'"');
  S:=New(PMemoryStream,Init(100,10));
  S^.StrWrite(P);
  S^.Seek(0);
  P:=Nil;
  P:=S^.StrRead;
  Dispose (S,Done);
  Writeln ('Read from stream : "',P,'"');
  Freemem(P,Strlen(P)+1);
end.

```

67.21.6 TStream.GetPos

Synopsis: Return current position in the stream.

Declaration: `function GetPos : LongInt; Virtual`

Visibility: default

Description: If the stream's status is `stOk`, `GetPos` returns the current position in the stream. Otherwise it returns `-1`

Errors: `-1` is returned if the status is an error condition.

See also: `TStream.Seek` ([1265](#)), `TStream.GetSize` ([1261](#))

Listing: `./examples/objectex/ex11.pp`

Program ex11;

{ Program to demonstrate the TStream.GetPos function }

Uses objects;

Var L : **String**;
 S : PStream;

```

begin
  L:='Some kind of string';
  S:=New(PMemoryStream,Init(100,10));
  Writeln ('Stream position before write: ',S^.GetPos);
  S^.WriteStr(@L);
  Writeln ('Stream position after write : ',S^.GetPos);
  Dispose(S,Done);
end.

```

67.21.7 TStream.GetSize

Synopsis: Return the size of the stream.

Declaration: `function GetSize : LongInt; Virtual`

Visibility: default

Description: If the stream's status is `stOk` then `GetSize` returns the size of the stream, otherwise it returns `-1`.

Errors: `-1` is returned if the status is an error condition.

See also: `TStream.Seek` ([1265](#)), `TStream.GetPos` ([1261](#))

Listing: `./examples/objectex/ex12.pp`

Program `ex12`;

{ Program to demonstrate the TStream.GetSize function }

Uses `objects`;

Var `L : String`;
 `S : PStream`;

begin
 `L := 'Some kind of string';`
 `S := New(PMemoryStream, Init(100, 10));`
 `Writeln ('Stream size before write: ', S^.GetSize);`
 `S^.WriteStr(@L);`
 `Writeln ('Stream size after write: ', S^.GetSize);`
 `Dispose(S, Done);`
end.

67.21.8 TStream.ReadStr

Synopsis: Read a shortstring from the stream.

Declaration: `function ReadStr : PString`

Visibility: default

Description: `ReadStr` reads a string from the stream, copies it to the heap and returns a pointer to this copy. The string is saved as a pascal string, and hence is NOT null terminated.

Errors: On error (e.g. not enough memory), `Nil` is returned.

See also: `TStream.StrRead` ([1260](#))

Listing: `./examples/objectex/ex13.pp`

Program `ex13`;

{
Program to demonstrate the TStream.ReadStr TStream.WriteStr functions
}

Uses `objects`;

Var `P : PString`;
 `L : String`;
 `S : PStream`;

begin

```

L:='Constant string line';
Writeln ('Writing to stream : "',L,'"');
S:=New(PMemoryStream,Init(100,10));
S^.WriteStr(@L);
S^.Seek(0);
P:=S^.ReadStr;
L:=P^;
DisposeStr(P);
Dispose (S,Done);
Writeln ('Read from stream : "',L,'"');
end.

```

67.21.9 TStream.Open

Synopsis: Open the stream.

Declaration: `procedure Open(OpenMode: Word); Virtual`

Visibility: default

Description: `Open` is an abstract method, that should be overridden by descendent objects. Since opening a stream depends on the stream's type this is not surprising.

For an example, see `TDosStream.Open` ([1235](#)).

Errors: None.

See also: `TStream.Close` ([1263](#)), `TStream.Reset` ([1263](#))

67.21.10 TStream.Close

Synopsis: Close the stream.

Declaration: `procedure Close; Virtual`

Visibility: default

Description: `Close` is an abstract method, that should be overridden by descendent objects. Since Closing a stream depends on the stream's type this is not surprising.

for an example, see `TDosStream.Open` ([1235](#)).

Errors: None.

See also: `TStream.Open` ([1263](#)), `TStream.Reset` ([1263](#))

67.21.11 TStream.Reset

Synopsis: Reset the stream.

Declaration: `procedure Reset`

Visibility: default

Description: `Reset` sets the stream's status to 0, as well as the `ErrorInfo`

Errors: None.

See also: `TStream.Open` ([1263](#)), `TStream.Close` ([1263](#))

67.21.12 TStream.Flush

Synopsis: Flush the stream data from the buffer, if any.

Declaration: `procedure Flush; Virtual`

Visibility: default

Description: `Flush` is an abstract method that should be overridden by descendent objects. It serves to enable the programmer to tell streams that implement a buffer to clear the buffer.

for an example, see `TBufStream.Flush` ([1215](#)).

Errors: None.

See also: `TStream.Truncate` ([1264](#))

67.21.13 TStream.Truncate

Synopsis: Truncate the stream size on current position.

Declaration: `procedure Truncate; Virtual`

Visibility: default

Description: `Truncate` is an abstract procedure that should be overridden by descendent objects. It serves to enable the programmer to truncate the size of the stream to the current file position.

For an example, see `TDosStream.Truncate` ([1234](#)).

Errors: None.

See also: `TStream.Seek` ([1265](#))

67.21.14 TStream.Put

Synopsis: Write an object to the stream.

Declaration: `procedure Put (P: PObject)`

Visibility: default

Description: `Put` writes the object pointed to by `P`. `P` should be non-nil. The object type must have been registered with `RegisterType` ([1210](#)).

After the object has been written, it can be read again with `Get` ([1259](#)).

For an example, see `TStream.Get` ([1259](#));

Errors: No check is done whether `P` is `Nil` or not. Passing `Nil` will cause a run-time error 216 to be generated. If the object has not been registered, the status of the stream will be set to `stPutError`.

See also: `TStream.Get` ([1259](#))

67.21.15 TStream.StrWrite

Synopsis: Write a null-terminated string to the stream.

Declaration: `procedure StrWrite(P: PChar)`

Visibility: default

Description: `StrWrite` writes the null-terminated string `P` to the stream. `P` can only be 65535 bytes long.

For an example, see `TStream.StrRead` ([1260](#)).

Errors: None.

See also: `TStream.WriteString` ([1265](#)), `TStream.StrRead` ([1260](#)), `TStream.ReadStr` ([1262](#))

67.21.16 TStream.WriteString

Synopsis: Write a pascal string to the stream.

Declaration: `procedure WriteStr(P: PString)`

Visibility: default

Description: `StrWrite` writes the pascal string pointed to by `P` to the stream.

For an example, see `TStream.ReadStr` ([1262](#)).

Errors: None.

See also: `TStream.StrWrite` ([1265](#)), `TStream.StrRead` ([1260](#)), `TStream.ReadStr` ([1262](#))

67.21.17 TStream.Seek

Synopsis: Set stream position.

Declaration: `procedure Seek(Pos: LongInt); Virtual`

Visibility: default

Description: `Seek` sets the position to `Pos`. This position is counted from the beginning, and is zero based. (i.e. `seek(0)` sets the position pointer on the first byte of the stream)

For an example, see `TDosStream.Seek` ([1234](#)).

Errors: If `Pos` is larger than the stream size, `Status` is set to `StSeekError`.

See also: `TStream.GetPos` ([1261](#)), `TStream.GetSize` ([1261](#))

67.21.18 TStream.Error

Synopsis: Set stream status.

Declaration: `procedure Error(Code: Integer; Info: Integer); Virtual`

Visibility: default

Description: `Error` sets the stream's status to `Code` and `ErrorInfo` to `Info`. If the `StreamError` procedural variable is set, `Error` executes it, passing `Self` as an argument.

This method should not be called directly from a program. It is intended to be used in descendent objects.

Errors: None.

67.21.19 TStream.Read

Synopsis: Read data from stream to buffer.

Declaration: `procedure Read(var Buf; Count: LongInt); Virtual`

Visibility: default

Description: Read is an abstract method that should be overridden by descendent objects.

Read reads Count bytes from the stream into Buf. It updates the position pointer, increasing it's value with Count. Buf must be large enough to contain Count bytes.

Errors: No checking is done to see if Buf is large enough to contain Count bytes.

See also: TStream.Write (1266), TStream.ReadStr (1262), TStream.StrRead (1260)

Listing: ./examples/objectex/ex18.pp

```

program ex18;

{ Program to demonstrate the TStream.Read method }

Uses Objects;

Var Buf1, Buf2 : Array[1..1000] of Byte;
    I : longint;
    S : PMemoryStream;

begin
    For I:=1 to 1000 do
        Buf1[I]:=Random(1000);
        Buf2:=Buf1;
        S:=New(PMemoryStream, Init(100,10));
        S^.Write(Buf1, SizeOf(Buf1));
        S^.Seek(0);
        For I:=1 to 1000 do
            Buf1[I]:=0;
            S^.Read(Buf1, SizeOf(Buf1));
            For I:=1 to 1000 do
                if Buf1[I]<>Buf2[I] then
                    WriteLn ('Buffer differs at position ', I);
            Dispose(S, Done);
end.

```

67.21.20 TStream.Write

Synopsis: Write a number of bytes to the stream.

Declaration: `procedure Write(var Buf; Count: LongInt); Virtual`

Visibility: default

Description: Write is an abstract method that should be overridden by descendent objects.

Write writes Count bytes to the stream from Buf. It updates the position pointer, increasing it's value with Count.

For an example, see TStream.Read (1266).

Errors: No checking is done to see if Buf actually contains Count bytes.

See also: TStream.Read (1266), TStream.WriteString (1265), TStream.StrWrite (1265)

67.21.21 TStream.CopyFrom

Synopsis: Copy data from another stream.

Declaration: `procedure CopyFrom(var S: TStream; Count: LongInt)`

Visibility: default

Description: `CopyFrom` reads `Count` bytes from stream `S` and stores them in the current stream. It uses the `Read` (1266) method to read the data, and the `Write` (1266) method to write in the current stream.

Errors: None.

See also: `Read` (1266), `Write` (1266)

Listing: `./examples/objectex/ex19.pp`

Program `ex19`;

{ Program to demonstrate the TStream.CopyFrom function }

Uses `objects`;

Var `P` : `PString`;
 `L` : **String**;
 `S1,S2` : `PStream`;

begin
 `L` := 'Constant string line';
 Writeln ('Writing to stream 1 : "',`L`,'"");
 `S1` := **New**(`PMemoryStream`,`Init`(100,10));
 `S2` := **New**(`PMemoryStream`,`Init`(100,10));
 `S1`^.`WriteStr`(`@L`);
 `S1`^.**Seek**(0);
 Writeln ('Copying contents of stream 1 to stream 2');
 `S2`^.`Copyfrom`(`S1`^,`S1`^.`GetSize`);
 `S2`^.**Seek**(0);
 `P` := `S2`^.`ReadStr`;
 `L` := `P`^;
 DisposeStr(`P`);
 Dispose (`S1`,`Done`);
 Dispose (`S2`,`Done`);
 Writeln ('Read from stream 2 : "',`L`,'"");
end.

67.22 TStringCollection**67.22.1 Description**

The `TStringCollection` object manages a sorted collection of pascal strings. To this end, it overrides the `Compare` (1253) method of `TSortedCollection`, and it introduces methods to read/write strings from a stream.

67.22.2 Method overview

Page	Method	Description
1268	Compare	Compare two strings in the collection.
1269	FreeItem	Dispose a string in the collection from memory.
1268	GetItem	Get string from the stream.
1269	PutItem	Write a string to the stream.

67.22.3 TStringCollection.GetItem

Synopsis: Get string from the stream.

Declaration: `function GetItem(var S: TStream) : Pointer; Virtual`

Visibility: default

Description: `GetItem` reads a string from the stream `S` and returns a pointer to it. It doesn't insert the string in the collection.

This method is primarily introduced to be able to load and store the collection from and to a stream.

Errors: The errors returned are those of `TStream.ReadStr` ([1262](#)).

See also: `TStringCollection.PutItem` ([1269](#))

67.22.4 TStringCollection.Compare

Synopsis: Compare two strings in the collection.

Declaration: `function Compare(Key1: Pointer; Key2: Pointer) : Sw_Integer; Virtual`

Visibility: default

Description: `TStringCollection` overrides the `Compare` function so it compares the two keys as if they were pointers to strings. The compare is done case sensitive. It returns the following results:

- 1 if the first string is alphabetically earlier than the second string.
- 0 if the two strings are equal.
- 1 if the first string is alphabetically later than the second string.

Errors: None.

See also: `TSortedCollection.Compare` ([1253](#))

Listing: `./examples/objectex/ex37.pp`

Program `ex37`;

{ Program to demonstrate the TStringCollection.Compare method }

Uses `Objects`;

Var `C : PStringCollection;`
`S : String;`
`I : longint;`

begin
`Randomize;`

```

C:=New(PStringCollection,Init(120,10));
C^.Duplicates:=True; { Duplicates allowed }
Writeln ('Inserting 100 records at random places. ');
For I:=1 to 100 do
  begin
    Str(Random(100),S);
    S:='String with value '+S;
    C^.Insert(NewStr(S));
  end;
For I:=0 to 98 do
  With C^ do
    If Compare (At(I),At(I+1))=0 then
      Writeln ('Duplicate string found at position ',I);
Dispose(C,Done);
end.

```

67.22.5 TStringCollection.FreeItem

Synopsis: Dispose a string in the collection from memory.

Declaration: `procedure FreeItem(Item: Pointer); Virtual`

Visibility: default

Description: `TStringCollection` overrides `FreeItem` so that the string pointed to by `Item` is disposed from memory.

Errors: None.

See also: `TCollection.FreeItem` ([1228](#))

67.22.6 TStringCollection.PutItem

Synopsis: Write a string to the stream.

Declaration: `procedure PutItem(var S: TStream; Item: Pointer); Virtual`

Visibility: default

Description: `PutItem` writes the string pointed to by `Item` to the stream `S`.

This method is primarily used in the `Load` and `Store` methods, and should not be used directly.

Errors: Errors are those of `TStream.WriteString` ([1265](#)).

See also: `TStringCollection.GetItem` ([1268](#))

67.23 TStringList

67.23.1 Description

A `TStringList` object can be used to read a collection of strings stored in a stream. If you register this object with the `RegisterType` ([1210](#)) function, you cannot register the `TStrListMaker` object.

67.23.2 Method overview

Page	Method	Description
1270	Done	Clean up the instance.
1270	Get	Return a string by key name.
1270	Load	Load stringlist from stream.

67.23.3 TStringList.Load

Synopsis: Load stringlist from stream.

Declaration: `constructor Load(var S: TStream)`

Visibility: default

Description: The `Load` constructor reads the `TStringList` object from the stream `S`. It also reads the descriptions of the strings from the stream. The string descriptions are stored as an array of `TStrIndexrec` records, where each record describes a string on the stream. These records are kept in memory.

Errors: If an error occurs, a stream error is triggered.

See also: `TStringList.Done` ([1270](#))

67.23.4 TStringList.Done

Synopsis: Clean up the instance.

Declaration: `destructor Done; Virtual`

Visibility: default

Description: The `Done` destructor frees the memory occupied by the string descriptions, and destroys the object.

Errors: None.

See also: `Load` ([1270](#)), `TObject.Done` ([1240](#))

67.23.5 TStringList.Get

Synopsis: Return a string by key name.

Declaration: `function Get(Key: Sw_Word) : string`

Visibility: default

Description: `Get` reads the string with key `Key` from the list of strings on the stream, and returns this string. If there is no string with such a key, an empty string is returned.

Errors: If no string with key `Key` is found, an empty string is returned. A stream error may result if the stream doesn't contain the needed strings.

See also: `TStrListMaker.Put` ([1271](#))

67.24 TStrListMaker

67.24.1 Description

The `TStrListMaker` object can be used to generate a stream with strings, which can be read with the `TStringList` object. If you register this object with the `RegisterType` (1210) function, you cannot register the `TStringList` object.

67.24.2 Method overview

Page	Method	Description
1271	<code>Done</code>	Clean up the instance and free all related memory.
1271	<code>Init</code>	Instantiate a new instance of <code>TStrListMaker</code> .
1271	<code>Put</code>	Add a new string to the list with associated key.
1272	<code>Store</code>	Write the strings to the stream.

67.24.3 TStrListMaker.Init

Synopsis: Instantiate a new instance of `TStrListMaker`.

Declaration: `constructor Init (AStrSize: Sw_Word; AIndexSize: Sw_Word)`

Visibility: default

Description: The `Init` constructor creates a new instance of the `TstrListMaker` object. It allocates `AStrSize` bytes on the heap to hold all the strings you wish to store. It also allocates enough room for `AIndexSize` key description entries (of the type `TStrIndexrec`).

`AStrSize` must be large enough to contain all the strings you wish to store. If not enough memory is allocated, other memory will be overwritten. The same is true for `AIndexSize` : maximally `AIndexSize` strings can be written to the stream.

Errors: None.

See also: `TObject.Init` (1239), `TStrListMaker.Done` (1271)

67.24.4 TStrListMaker.Done

Synopsis: Clean up the instance and free all related memory.

Declaration: `destructor Done; Virtual`

Visibility: default

Description: The `Done` destructor de-allocates the memory for the index description records and the string data, and then destroys the object.

Errors: None.

See also: `TObject.Done` (1240), `TStrListMaker.Init` (1271)

67.24.5 TStrListMaker.Put

Synopsis: Add a new string to the list with associated key.

Declaration: `procedure Put (Key: Sw_Word; S: string)`

Visibility: default

Description: `Put` adds the string `S` with key `Key` to the collection of strings. This action doesn't write the string to a stream. To write the strings to the stream, see the `Store` ([1272](#)) method.

Errors: None.

See also: `TStrListMaker.Store` ([1272](#))

67.24.6 TStrListMaker.Store

Synopsis: Write the strings to the stream.

Declaration: `procedure Store(var S: TStream)`

Visibility: default

Description: `Store` writes the collection of strings to the stream `S`. The collection can then be read with the `TStringList` object.

Errors: A stream error may occur when writing the strings to the stream.

See also: `TStringList.Load` ([1270](#)), `TStrListMaker.Put` ([1271](#))

67.25 TUnSortedStrCollection

67.25.1 Description

The `TUnSortedStrCollection` object manages an unsorted list of strings. To this end, it overrides the `TSortedCollection.Insert` ([1255](#)) method to add strings at the end of the collection, rather than in the alphabetically correct position.

Take care, the `Search` ([1254](#)) and `IndexOf` ([1220](#)) methods will not work on an unsorted string collection.

67.25.2 Method overview

Page	Method	Description
1272	<code>Insert</code>	Insert a new string in the collection.

67.25.3 TUnSortedStrCollection.Insert

Synopsis: Insert a new string in the collection.

Declaration: `procedure Insert(Item: Pointer); Virtual`

Visibility: default

Description: `Insert` inserts a string at the end of the collection, instead of on its alphabetical place, resulting in an unsorted collection of strings.

Errors: None.

See also: `TCollection.Insert` ([1226](#))

Listing: `./examples/objectex/ex39.pp`

Program ex39;

{ Program to demonstrate the TUnsortedStrCollection.Insert method }

Uses Objects,Strings;

Var C : PUnsortedStrCollection;

 S : **String**;

 I : longint;

 P : Pchar;

begin

Randomize;

 C:=**New**(PUnsortedStrCollection,Init(120,10));

Writeln ('Inserting 100 records at random places.');

For I:=1 **to** 100 **do**

begin

Str(**Random**(100),S);

 S:='String with value '+S;

 C[^].**Insert**(**NewStr**(S));

end;

For I:=0 **to** 99 **do**

Writeln (I:2,': ',PString(C[^].**At**(i))[^]);

Dispose(C,Done);

end.

Chapter 68

Reference for unit 'objpas'

68.1 Used units

Table 68.1: Used units by unit 'objpas'

Name	Page
System	1335

68.2 Overview

The `objpas` unit is meant for compatibility with Object Pascal as implemented by Delphi. The unit is loaded automatically by the Free Pascal compiler whenever the `Delphi` or `objfpc` mode is entered, either through the command line switches `-Sd` or `-Sh` or with the `{ $MODE DELPHI }` or `{ $MODE OBJFPC }` directives.

It redefines some basic pascal types, introduces some functions for compatibility with Delphi's system unit, and introduces some methods for the management of the resource string tables.

68.3 Constants, types and variables

68.3.1 Constants

`MaxInt = MaxLongint`

Maximum value for Integer ([1275](#)) type.

68.3.2 Types

`FixedInt = Int32`

`FixedInt` is provided for Delphi compatibility.

`FixedUInt = UInt32`

FixedUInt is provided for Delphi compatibility.

Integer = LongInt

In OBJPAS mode and in DELPHI mode, an Integer has a size of 32 bit. In TP or regular FPC mode, an integer is 16 bit.

IntegerArray = Array[0..\$effffff] of Integer

Generic array of integer (1275).

PFixedInt = ^FixedInt

PFixedUInt = ^FixedUInt

PInteger = ^Integer

Pointer to Integer (1275) type.

PIntegerArray = ^IntegerArray

Pointer to TIntegerArray (1275) type.

PointerArray = Array[0..512*1024*1024-2] of Pointer

Generic Array of pointers.

PPointerArray = ^PointerArray

Pointer to PointerArray (1275).

PString = PAnsiString

Pointer to ansistring type.

TEndian = (Little, Big)

Table 68.2: Enumeration values for type TEndian

Value	Explanation
Big	Big endian byte order.
Little	Little endian byte order.

TEndian describes the endianness of a computer architecture.

TIntegerArray = IntegerArray

Alias for IntegerArray (1275).

TPointerArray = PointerArray

Alias for PointerArray (1275).

Chapter 69

Reference for unit 'ports'

69.1 Used units

Table 69.1: Used units by unit 'ports'

Name	Page
System	1335

69.2 Overview

The ports unit implements the `port` constructs found in Turbo Pascal. It uses classes and default array properties to do this.

The unit exists on Linux, OS/2 and Dos. It is implemented only for compatibility with Turbo Pascal. Its usage is discouraged, because using ports is not portable programming, and the operating system may not even allow it (for instance Windows).

Under Linux, your program must be run as root, or the `IOPerm` call must be set in order to set appropriate permissions on the port access.

69.3 Constants, types and variables

69.3.1 Variables

```
port : tport
```

Default instance of type `TPort` ([1277](#)). Do not free. This variable is initialized in the unit initialization code, and freed at finalization.

Since there is a default property for a variable of this type, a sentence as

```
port[221]:=12;
```

Will result in the integer 12 being written to port 221, if port is defined as a variable of type `tport`

```
portb : tport
```

Default instance of type `TPort` ([1277](#)). Do not free. This variable is initialized in the unit initialization code, and freed at finalization.

Since there is a default property for a variable of this type, a sentence as

```
portb[221]:=12;
```

Will result in the byte 12 being written to port 221, if port is defined as a variable of type `tport`

```
portl : tportl
```

Default instance of type `TPortL` ([1278](#)). Do not free. This variable is initialized in the unit initialization code, and freed at finalization.

Since there is a default property for a variable of this type, a sentence as

```
portl[221]:=12;
```

Will result in the longint 12 being written to port 221, if port is defined as a variable of type `tport`

```
portw : tportw
```

Default instance of type `TPortW` ([1278](#)). Do not free. This variable is initialized in the unit initialization code, and freed at finalization.

Since there is a default property for a variable of this type, a sentence as

```
portw[221]:=12;
```

Will result in the word 12 being written to port 221, if port is defined as a variable of type `tport`

69.4 tport

69.4.1 Description

The `TPort` type is implemented specially for access to the ports in a TP compatible manner. There is no need to create an instance of this type: the standard TP variables are instantiated at unit initialization.

See also: `port` ([1276](#)), `TPortW` ([1278](#)), `TPortL` ([1278](#))

69.4.2 Property overview

Page	Properties	Access	Description
1277	pp	rw	Access integer-sized port by port number.

69.4.3 tport.pp

Synopsis: Access integer-sized port by port number.

Declaration: `Property pp[w: Word]: Byte; default`

Visibility: `public`

Access: `Read,Write`

Description: Access integer-sized port by port number.

69.5 tportl

69.5.1 Description

The `TPortL` type is implemented specially for access to the ports in a TP compatible manner. There is no need to create an instance of this type: the standard TP variables are instantiated at unit initialization.

See also: `portw` ([1277](#)), `TPort` ([1277](#)), `TPortL` ([1278](#))

69.5.2 Property overview

Page	Properties	Access	Description
1278	pp	rw	Access Longint-sized port by port number.

69.5.3 tportl.pp

Synopsis: Access Longint-sized port by port number.

Declaration: `Property pp[w: Word]: LongInt; default`

Visibility: `public`

Access: `Read,Write`

Description: Access Longint-sized port by port number.

69.6 tportw

69.6.1 Description

The `TPortW` type is implemented specially for access to the ports in a TP compatible manner. There is no need to create an instance of this type: the standard TP variables are instantiated at unit initialization.

See also: `portw` ([1277](#)), `TPort` ([1277](#)), `TPortL` ([1278](#))

69.6.2 Property overview

Page	Properties	Access	Description
1278	pp	rw	Access word-sized port by port number.

69.6.3 tportw.pp

Synopsis: Access word-sized port by port number.

Declaration: `Property pp[w: Word]: Word; default`

Visibility: `public`

Access: `Read,Write`

Description: Access word-sized port by port number.

Chapter 70

Reference for unit 'printer'

70.1 Used units

Table 70.1: Used units by unit 'printer'

Name	Page
System	1335

70.2 Overview

This chapter describes the `printer` unit for Free Pascal. It was written for DOS by Florian Klaempfl, and it was written for Linux by Michael Van Canneyt, and has been ported to Windows and OS/2 as well. Its basic functionality is the same for all supported systems, although there are minor differences on Linux and UNIX.

70.3 Constants, types and variables

70.3.1 Variables

`Lst` : `text`

`Lst` is the standard printing device.

On Linux, `Lst` is set up using `AssignLst('/tmp/PID.lst')`.

70.4 Procedures and functions

70.4.1 AssignLst

Synopsis: Assign text file to printing device.

Declaration: `procedure AssignLst (var F: text; ToFile: string)`

Visibility: default

Description: `AssignLst` assigns to `F` a printing device - *UNIX only*. `ToFile` is a string with the following form:

- `'|filename options'`: This sets up a pipe with the program filename, with the given options, such as in the `popen()` call.
- `'filename'`: Prints to file filename. Filename can contain the string 'PID' (No Quotes), which will be replaced by the PID of your program. When closing `lst`, the file will be sent to `lpr` and deleted. (`lpr` should be in `PATH`)
- `{ 'filename| ' }`: Same as previous, only the file is *not* sent to `lpr`, nor is it deleted. (useful for opening `/dev/printer` or for later printing)

See also: `lst` ([1279](#))

Listing: `./examples/printex/printex.pp`

program testprn;

uses printer;

var i : integer;
f : text;

begin

writeln ('Test of printer unit');

writeln ('Writing to lst...');

for i:=1 **to** 80 **do** **writeln** (lst,'This is line ',i,'.#13');

close (lst);

writeln ('Done.');

 {\$ifdef Unix}

writeln ('Writing to pipe...');

assignlst (f,'/usr/bin/lpr -m');

rewrite (f);

for i:=1 **to** 80 **do** **writeln** (f,'This is line ',i,'.#13');

close (f);

writeln ('Done.')

 {\$endif}

end.

70.4.2 InitPrinter

Synopsis: Initialize the printer.

Declaration: `procedure InitPrinter(const PrinterName: string)`

Visibility: default

Description: Initialize the printer.

70.4.3 IsLstAvailable

Synopsis: Determine whether printer is available.

Declaration: `function IsLstAvailable : Boolean`

Visibility: default

Description: Determine whether printer is available.

Chapter 71

Reference for unit 'sharemem'

71.1 Used units

Table 71.1: Used units by unit 'sharemem'

Name	Page
System	1335

71.2 Overview

`sharemem` implements a shared memory manager. Including this unit will replace the standard memory manager with a memory manager which uses shared memory. This means the memory allocated by this unit can be managed by a program and a DLL if they both use the shared memory manager: it allows, amongst other things, to pass ansistrings or Unicode strings from a program to a DLL and vice versa.

This unit does not implement any routines: all actions to replace the memory manager are performed in the initialization section of the unit. The unit should be placed as the first unit in a program or DLL's uses section, memory corruption may occur if the unit is not placed first.

This unit requires the `fpcmemdll.dll` library to be distributed with both program and dll that use this unit. This DLL is distributed with the windows Free Pascal distribution.

Chapter 72

Reference for unit 'Sockets'

72.1 Used units

Table 72.1: Used units by unit 'Sockets'

Name	Page
BaseUnix	155
System	1335
unixtype	2229

72.2 Overview

This document describes the SOCKETS unit for Free Pascal. it was written for Linux by Michael Van Canneyt, and ported to Windows by Florian Klaempfl.

Chapter 73

Reference for unit 'Strings'

73.1 Used units

Table 73.1: Used units by unit 'Strings'

Name	Page
System	1335

73.2 Overview

This chapter describes the `STRINGS` unit for Free Pascal. This unit is system independent, and therefore works on all supported platforms.

73.3 Procedures and functions

73.3.1 `stralloc`

Synopsis: Allocate memory for a new null-terminated string on the heap.

Declaration: `function stralloc(L: SizeInt) : PChar`

Visibility: default

Description: `StrAlloc` reserves memory on the heap for a string with length `Len`, terminating `#0` included, and returns a pointer to it.

Errors: If there is not enough memory, a run-time error occurs.

See also: `StrNew` ([1292](#)), `StrPCopy` ([1293](#))

73.3.2 `strcat`

Synopsis: Concatenate 2 null-terminated strings.

Declaration: `function strcat(dest: PChar; source: PChar) : PChar`

Visibility: default

Description: Attaches Source to Dest and returns Dest.

Errors: No length checking is performed.

See also: StrLCat ([1288](#))

Listing: ./examples/stringex/ex11.pp

Program Example11;

Uses strings;

{ Program to demonstrate the StrCat function. }

Const P1 : PChar = 'This is a PChar String.';

Var P2 : PChar;

begin

P2:=StrAlloc (StrLen(P1)*2+1);

StrMove (P2,P1,StrLen(P1)+1); { P2=P1 }

StrCat (P2,P1); { Append P2 once more }

Writeln ('P2 : ',P2);

StrDispose(P2);

end.

73.3.3 strcmp

Synopsis: Compare 2 null-terminated strings, case sensitive.

Declaration: function strcmp(str1: PChar; str2: PChar) : SizeInt

Visibility: default

Description: Compares the null-terminated strings S1 and S2. The result is

- A negative SizeInt when S1<S2.
- 0 when S1=S2.
- A positive SizeInt when S1>S2.

For an example, see StrLComp ([1289](#)).

Errors: None.

See also: StrLComp ([1289](#)), StrIComp ([1287](#)), StrLComp ([1290](#))

73.3.4 strcpy

Synopsis: Copy a null-terminated string.

Declaration: function strcpy(dest: PChar; source: PChar) : PChar; Overload

Visibility: default

Description: Copy the null terminated string in Source to Dest, and returns a pointer to Dest. Dest needs enough room to contain Source, i.e. StrLen (Source) +1 bytes.

Errors: No length checking is performed.

See also: `StrPCopy` ([1293](#)), `StrLCopy` ([1289](#)), `StrECopy` ([1286](#))

Listing: `./examples/stringex/ex4.pp`

Program Example4;

Uses strings;

{ Program to demonstrate the StrCopy function. }

Const P : PChar = 'This is a PCHAR string.';

var PP : PChar;

begin

PP:=StrAlloc(Strlen(P)+1);

StrCopy (PP,P);

If StrComp (PP,P)<>0 **then**

Writeln ('Oh-oh problems...')

else

Writeln ('All is well : PP=',PP);

StrDispose(PP);

end.

73.3.5 strdispose

Synopsis: disposes of a null-terminated string on the heap.

Declaration: `procedure strdispose(p: PChar)`

Visibility: default

Description: Removes the string in P from the heap and releases the memory.

Errors: None.

See also: `StrNew` ([1292](#))

Listing: `./examples/stringex/ex17.pp`

Program Example17;

Uses strings;

{ Program to demonstrate the StrDispose function. }

Const P1 : PChar = 'This is a PChar string';

var P2 : PChar;

begin

P2:=StrNew (P1);

Writeln ('P2 : ',P2);

StrDispose(P2);

end.

73.3.6 strecopy

Synopsis: Copy a null-terminated string, return a pointer to the end.

Declaration: `function strecopy(dest: PChar; source: PChar) : PChar`

Visibility: default

Description: Copies the Null-terminated string in `Source` to `Dest`, and returns a pointer to the end (i.e. the terminating Null-character) of the copied string.

Errors: No length checking is performed.

See also: `StrLCopy` ([1289](#)), `StrCopy` ([1284](#))

Listing: `./examples/stringex/ex6.pp`

Program Example6;

Uses strings;

{ Program to demonstrate the StrECopy function. }

Const P : PChar = 'This is a PCHAR string.';

Var PP : PChar;

begin

PP:=StrAlloc (StrLen(P)+1);

If Longint(StrECopy(PP,P))-Longint(PP)<>StrLen(P) **then**

 Writeln('Something is wrong here !')

else

 Writeln ('PP= ',PP);

 StrDispose(PP);

end.

73.3.7 strend

Synopsis: Return a pointer to the end of a null-terminated string.

Declaration: `function strend(p: PChar) : PChar`

Visibility: default

Description: Returns a pointer to the end of P. (i.e. to the terminating null-character.

Errors: None.

See also: `StrLen` ([1290](#))

Listing: `./examples/stringex/ex7.pp`

Program Example6;

Uses strings;

{ Program to demonstrate the StrEnd function. }

Const P : PChar = 'This is a PCHAR string.';

```

begin
  If Longint(StrEnd(P))-Longint(P)<>StrLen(P) then
    Writeln('Something is wrong here !')
  else
    Writeln ('All is well..');
end.

```

73.3.8 stricmp

Synopsis: Compare 2 null-terminated strings, case insensitive.

Declaration: `function stricmp(str1: PChar; str2: PChar) : SizeInt`

Visibility: default

Description: Compares the null-terminated strings S1 and S2, ignoring case. The result is

- A negative `SizeInt` when $S1 < S2$.
- 0 when $S1 = S2$.
- A positive `SizeInt` when $S1 > S2$.

Errors: None.

See also: `StrLComp` ([1289](#)), `StrComp` ([1284](#)), `StrLComp` ([1290](#))

Listing: `./examples/stringex/ex8.pp`

Program Example8;

Uses strings;

{ Program to demonstrate the StrLComp function. }

Const P1 : PChar = 'This is the first string.';
 P2 : PChar = 'This is the second string.';

Var L : Longint;

```

begin
  Write ('P1 and P2 are ');
  If StrComp (P1,P2)<>0 then write ('NOT ');
  write ('equal. The first ');
  L:=1;
  While StrLComp(P1,P2,L)=0 do inc (L);
  dec(L);
  Writeln (L,' characters are the same.');
```

end.

73.3.9 stripos

Synopsis: Return the position of a substring in a string, case insensitive.

Declaration: `function stripos(str1: PChar; str2: PChar) : PChar`

Visibility: default

Description: `strpos` returns the position of `str2` in `str1`. It searches in a case-insensitive manner, and if it finds a match, it returns a pointer to the location of the match. If no match is found, `Nil` is returned.

Errors: No checks are done on the validity of the pointers, and the pointers are assumed to point to a properly null-terminated string. If either of these conditions are not met, a run-time error may follow.

See also: `strscan` (1288), `strpos` (1294)

73.3.10 `strscan`

Synopsis: Scan a string for a character, case-insensitive.

Declaration: `function strscan(p: PChar; c: Char) : PChar`

Visibility: default

Description: `strscan` does the same as `strscan` (1295) but compares the characters case-insensitively. It returns a pointer to the first occurrence of the character `c` in the null-terminated string `p`, or `Nil` if `c` is not present in the string.

See also: `strscan` (1295), `strrscan` (1294)

73.3.11 `strlcat`

Synopsis: Concatenate 2 null-terminated strings, with length boundary.

Declaration: `function strlcat(dest: PChar; source: PChar; l: SizeInt) : PChar`

Visibility: default

Description: Adds `L` characters from `Source` to `Dest`, and adds a terminating null-character. Returns `Dest`.

Errors: None.

See also: `StrCat` (1283)

Listing: `./examples/stringex/ex12.pp`

Program Example12;

Uses strings;

{ Program to demonstrate the StrLCat function. }

Const P1 : PChar = '1234567890';

Var P2 : PChar;

begin

 P2:=StrAlloc (StrLen(P1)*2+1);

 P2^:=#0; { Zero length }

 StrCat (P2,P1);

 StrLCat (P2,P1,5);

 Writeln ('P2 = ',P2);

 StrDispose(P2)

end.

73.3.12 strlcomp

Synopsis: Compare limited number of characters of 2 null-terminated strings.

Declaration: `function strlcomp(str1: PChar; str2: PChar; l: SizeInt) : SizeInt`

Visibility: default

Description: Compares maximum L characters of the null-terminated strings S1 and S2. The result is

- A negative `SizeInt` when `S1<S2`.
- 0 when `S1=S2`.
- A positive `SizeInt` when `S1>S2`.

Errors: None.

See also: `StrComp` ([1284](#)), `StrIComp` ([1287](#)), `StrLComp` ([1290](#))

Listing: `./examples/stringex/ex8.pp`

Program Example8;

Uses strings;

{ Program to demonstrate the StrLComp function. }

Const P1 : PChar = 'This is the first string.';
P2 : PChar = 'This is the second string.';

Var L : Longint;

begin

Write ('P1 and P2 are ');
If `StrComp` (P1,P2)<>0 **then write** ('NOT ');
write ('equal. The first ');
L:=1;
While `StrLComp`(P1,P2,L)=0 **do inc** (L);
dec(L);
Writeln (L,' characters are the same.');

end.

73.3.13 strlcopy

Synopsis: Copy a null-terminated string, limited in length.

Declaration: `function strlcopy(dest: PChar; source: PChar; maxlen: SizeInt) : PChar`
; Overload

Visibility: default

Description: Copies MaxLen characters from Source to Dest, and makes Dest a null terminated string.

Errors: No length checking is performed.

See also: `StrCopy` ([1284](#)), `StrECopy` ([1286](#))

Listing: `./examples/stringex/ex5.pp`

Program Example5;

Uses strings;

{ Program to demonstrate the StrLCopy function. }

Const P : PChar = '123456789ABCDEF';

var PP : PChar;

begin

PP:=StrAlloc(11);

Writeln ('First 10 characters of P : ',StrLCopy (PP,P,10));

StrDispose(PP);

end.

73.3.14 strlen

Synopsis: Length of a null-terminated string.

Declaration: `function strlen(p: PChar) : SizeInt`

Visibility: default

Description: Returns the length of the null-terminated string P. If P equals Nil, then zero (0) is returned.

Errors: None.

See also: StrNew ([1292](#))

Listing: ./examples/stringex/ex1.pp

Program Example1;

Uses strings;

{ Program to demonstrate the StrLen function. }

Const P : PChar = 'This is a constant pchar string';

begin

Writeln ('P : ',p);

Writeln ('length(P) : ',StrLen(P));

end.

73.3.15 strlicomp

Synopsis: Compare limited number of characters in 2 null-terminated strings, ignoring case.

Declaration: `function strlicomp(str1: PChar; str2: PChar; l: SizeInt) : SizeInt`

Visibility: default

Description: Compares maximum L characters of the null-terminated strings S1 and S2, ignoring case. The result is

- A negative `SizeInt` when $S1 < S2$.
- 0 when $S1 = S2$.
- A positive `SizeInt` when $S1 > S2$.

For an example, see `StrIComp` ([1287](#))

Errors: None.

See also: `StrLComp` ([1289](#)), `StrComp` ([1284](#)), `StrIComp` ([1287](#))

73.3.16 `strlower`

Synopsis: Convert null-terminated string to all-lowercase.

Declaration: `function strlower(p: PChar) : PChar`

Visibility: default

Description: Converts `P` to an all-lowercase string. Returns `P`.

Errors: None.

See also: `StrUpper` ([1295](#))

Listing: `./examples/stringex/ex14.pp`

Program Example14;

Uses strings;

{ Program to demonstrate the StrLower and StrUpper functions. }

Const

`P1 : PChar = 'THIS IS AN UPPERCASE PCHAR STRING';`
`P2 : PChar = 'this is a lowercase string';`

begin

`P1 := StrNew(P1);`
`P2 := strNew(P2);`
`Writeln ('Uppercase : ', StrUpper(P2));`
`StrLower(P1);`
`Writeln ('Lowercase : ', P1);`
`StrDispose(P1);`
`StrDispose(P2);`

end.

73.3.17 `strmove`

Synopsis: Move a null-terminated string to new location.

Declaration: `function strmove(dest: PChar; source: PChar; l: SizeInt) : PChar`

Visibility: default

Description: Copies `MaxLen` characters from `Source` to `Dest`. No terminating null-character is copied. Returns `Dest`

Errors: None.

See also: StrLCopy ([1289](#)), StrCopy ([1284](#))

Listing: ./examples/stringex/ex10.pp

Program Example10;

Uses strings;

{ Program to demonstrate the StrMove function. }

Const P1 : PCHAR = 'This is a pchar string.';

Var P2 : Pchar;

begin

 P2:=StrAlloc(StrLen(P1)+1);

StrMove (P2,P1,StrLen(P1)+1); { P2:=P1 }

Writeln ('P2 = ',P2);

StrDispose(P2);

end.

73.3.18 strnew

Synopsis: Allocate room for new null-terminated string.

Declaration: `function strnew(p: PChar) : PChar`

Visibility: default

Description: Copies P to the Heap, and returns a pointer to the copy.

Errors: Returns Nil if no memory was available for the copy.

See also: StrCopy ([1284](#)), StrDispose ([1285](#))

Listing: ./examples/stringex/ex16.pp

Program Example16;

Uses strings;

{ Program to demonstrate the StrNew function. }

Const P1 : PChar = 'This is a PChar string';

var P2 : PChar;

begin

 P2:=StrNew (P1);

If P1=P2 **then**

writeln ('This can"t be happening...')

else

writeln ('P2 : ',P2);

StrDispose(P2);

end.

73.3.19 strpas

Synopsis: Convert a null-terminated string to a shortstring.

Declaration: `function strpas(p: PChar) : shortstring`

Visibility: default

Description: Converts a null terminated string in P to a Pascal string, and returns this string. The string is truncated at 255 characters.

Errors: None.

See also: StrPCopy ([1293](#))

Listing: ./examples/stringex/ex3.pp

Program Example3;

Uses strings;

{ Program to demonstrate the StrPas function. }

Const P : PChar = 'This is a PCHAR string';

var S : string;

begin

 S:=StrPas (P);

 Writeln ('S : ',S);

end.

73.3.20 strcpy

Synopsis: Copy a pascal string to a null-terminated string.

Declaration: `function strcpy(d: PChar; const s: string) : PChar`

Visibility: default

Description: Converts the Pascal string in Se to a Null-terminated string, and copies it to D. D needs enough room to contain the string Source, i.e. Length (S) +1 bytes.

Errors: No length checking is performed.

See also: StrPas ([1293](#))

Listing: ./examples/stringex/ex2.pp

Program Example2;

Uses strings;

{ Program to demonstrate the StrPCopy function. }

Const S = 'This is a normal string.';

Var P : Pchar;

```

begin
  p:=StrAlloc (length(S)+1);
  if StrPCopy (P,S)<>P then
    Writeln ('This is impossible !!')
  else
    writeln (P);
    StrDispose(P);
end.

```

73.3.21 strpos

Synopsis: Search for a null-terminated substring in a null-terminated string.

Declaration: `function strpos(str1: PChar; str2: PChar) : PChar`

Visibility: default

Description: Returns a pointer to the first occurrence of S2 in S1. If S2 does not occur in S1, returns Nil.

Errors: None.

See also: StrScan ([1295](#)), StrRScan ([1295](#))

Listing: ./examples/stringex/ex15.pp

Program Example15;

Uses strings;

{ Program to demonstrate the StrPos function. }

Const P : PChar = 'This is a PChar string.';
 S : Pchar = 'is';

```

begin
  Writeln ('Position of "is" in P : ',sizeint(StrPos(P,S))-sizeint(P));
end.

```

73.3.22 striscan

Synopsis: Scan a string reversely for a character, case-insensitive.

Declaration: `function striscan(p: PChar; c: Char) : PChar`

Visibility: default

Description: `striscan` does the same as `strrscan` ([1295](#)) but compares the characters case-insensitively. It returns a pointer to the last occurrence of the character `c` in the null-terminated string `p`, or Nil if `c` is not present in the string.

See also: `strrscan` ([1295](#)), `striscan` ([1288](#))

73.3.23 strscan

Synopsis: Find last occurrence of a character in a null-terminated string.

Declaration: `function strscan(p: PChar; c: Char) : PChar`

Visibility: default

Description: Returns a pointer to the last occurrence of the character C in the null-terminated string P. If C does not occur, returns Nil.

For an example, see StrScan ([1295](#)).

Errors: None.

See also: StrScan ([1295](#)), StrPos ([1294](#))

73.3.24 strscan

Synopsis: Find first occurrence of a character in a null-terminated string.

Declaration: `function strscan(p: PChar; c: Char) : PChar`

Visibility: default

Description: Returns a pointer to the first occurrence of the character C in the null-terminated string P. If C does not occur, returns Nil.

Errors: None.

See also: StrRScan ([1295](#)), StrPos ([1294](#))

Listing: ./examples/stringex/ex13.pp

Program Example13;

Uses strings;

{ Program to demonstrate the StrScan and StrRScan functions. }

Const P : PChar = 'This is a PCHAR string.';
S : Char = 's' ;

begin

Writeln ('P, starting from first "s" : ',StrScan(P,s));

Writeln ('P, starting from last "s" : ',StrRScan(P,s));

end.

73.3.25 strupper

Synopsis: Convert null-terminated string to all-uppercase.

Declaration: `function strupper(p: PChar) : PChar`

Visibility: default

Description: Converts P to an all-uppercase string. Returns P.

For an example, see StrLower ([1291](#))

Errors: None.

See also: StrLower ([1291](#))

Chapter 74

Reference for unit 'StrUtils'

74.1 Used units

Table 74.1: Used units by unit 'StrUtils'

Name	Page
System	1335
sysutils	1609
Types	2007

74.2 Constants, types and variables

74.2.1 Resource strings

```
SErrAmountStrings =  
    'Amount of search and replace strings don''t match'
```

Error message used in stringsreplace function.

```
SInvalidRomanNumeral = '%s is not a valid Roman numeral'
```

Error string shown in exception raised when invalid roman numeral is encountered.

74.2.2 Constants

```
AnsiResemblesProc : TCompareTextProc = @ SoundexProc
```

This procedural variable is standard set to SoundexProc ([1328](#)) but can be overridden with a user-defined algorithm. This algorithm should return `True` if `AText` resembles `AOtherText`, or `False` otherwise. The standard routine compares the soundexes of the two strings and returns `True` if they are equal.

```
Brackets = ['(', ')', '[', ']', '{', '}']
```

Set of characters that contain all possible bracket characters.

```
DigitChars = ['0'..'9']
```

Set of digit characters.

```
ResemblesProc : TCompareTextProc = @ SoundExProc
```

`ResemblesProc` is a procedure variable which can be set to compare 2 string. It is similar to `AnsiResemblesProc` (1296), and by default is set to the `SoundExProc` (1328) function.

```
StdSwitchChars = ['-','/']
```

Standard characters for the `SwitchChars` argument of `GetCmdLineArg` (1312).

```
StdWordDelims = [#0..' ', ',', '.', ';', '/', '\', ':', "'", '"',
  , '`'] + Brackets
```

Standard word delimiter values.

```
WordDelimiters : Set of AnsiChar = [#0..#255] - ['a'..'z', 'A'..'Z',
  , '1'..'9', '0']
```

Standard word delimiters, used in the `SearchBuf` (1326) call.

74.2.3 Types

```
SizeIntArray = Array of SizeInt
```

`SizeIntArray` is an auxiliary type used in `StringReplace` (1330)

```
TCompareTextProc = function(const AText: string; const AOther: string
  )
  : Boolean
```

Function prototype for comparing two string in `AnsiResemblesText` (1303).

```
TRawByteStringArray = Array of RawByteString
```

`TRawByteStringArray` defines a dynamic array of `RawByteString`. It is used in the `RawByteString` version of the `SplitCommandLine` (1329) function.

```
TRomanConversionStrictness = (rcsStrict, rcsRelaxed, rcsDontCare)
```

Table 74.2: Enumeration values for type `TRomanConversionStrictness`

Value	Explanation
<code>rcsDontCare</code>	Do not check correctness.
<code>rcsRelaxed</code>	Like <code>rcsStrict</code> but allow more than 3 consecutive identical letters.
<code>rcsStrict</code>	Only accept correct roman numerals.

`TRomanConversionStrictness` is an enumerated type that can be used to decide how to react to invalid roman numerals.

rcsStrict Strict adherence to roman numerals. Up to 3 consecutive identical letters. No negative numbers. Ordering must be correct.

rcsRelaxed Same as `rcsStrict` but allow more than 3 consecutive identical letters.

rcsDontCare Do not check validity at all

`TSoundexIntLength = 1..8`

Range of allowed integer soundex lengths.

`TSoundexLength = 1..MaxInt`

Range of allowed soundex lengths.

`TStringReplaceAlgorithm = (sraDefault, sraManySmall, sraBoyerMoore)`

Table 74.3: Enumeration values for type `TStringReplaceAlgorithm`

Value	Explanation
<code>sraBoyerMoore</code>	Use a Boyer-Moore search algorithm.
<code>sraDefault</code>	Use the <code>sysutils</code> algorithm, which does a straightforward linear search and replace.
<code>sraManySmall</code>	Use an approach which is suitable for a string with many occurrences of the same small text.

`TStringReplaceAlgorithm` enumerates the available algorithms to `StringReplace` ([1330](#))

sraDefault Use the `sysutils` algorithm, which does a straightforward linear search and replace.

sraManySmall Use an approach which is suitable for a string with many occurrences of the same small text.

sraBoyerMoore Use a Boyer-Moore search algorithm.

Depending on the kind of data that is being treated, one or the other of these algorithms may produce faster results.

`TStringSeachOption = TStringSearchOption`

There is a typo error in the original Borland `StrUtils` unit. This type just refers to the correct `TStringSearchOption` ([1298](#)) and is provided for compatibility only.

`TStringSearchOption = (soDown, soMatchCase, soWholeWord)`

Table 74.4: Enumeration values for type `TStringSearchOption`

Value	Explanation
<code>soDown</code>	Search in down direction.
<code>soMatchCase</code>	Match case.
<code>soWholeWord</code>	Search whole words only.

Possible options for `SearchBuf` ([1326](#)) call.

TStringSearchOptions = Set of TStringSearchOption

Set of options for SearchBuf ([1326](#)) call.

TUnicodeStringArray = Array of UnicodeString

TUnicodeStringArray defines a dynamic array of UnicodeString. It is used in the UnicodeString version of the SplitCommandLine ([1329](#)) function.

74.3 Procedures and functions

74.3.1 AddChar

Synopsis: Add characters to the left of a string till a certain length.

Declaration: `function AddChar(C: AnsiChar; const S: string; N: Integer) : string`

Visibility: default

Description: AddChar adds characters (C) to the left of S till the length N is reached, and returns the resulting string. If the length of S is already equal to or larger than N, then no characters are added. The resulting string can be thought of as a right-aligned version of S, with length N.

Errors: None

See also: AddCharR ([1299](#)), PadLeft ([1320](#)), PadRight ([1321](#)), PadCenter ([1320](#))

74.3.2 AddCharR

Synopsis: Add chars at the end of a string till it reaches a certain length.

Declaration: `function AddCharR(C: AnsiChar; const S: string; N: Integer) : string`

Visibility: default

Description: AddCharR adds characters (C) to the right of S till the length N is reached, and returns the resulting string. If the length of S is already equal to or larger than N, then no characters are added. The resulting string can be thought of as a left-aligned version of S, with length N.

Errors: None

See also: AddChar ([1299](#))

74.3.3 AnsiContainsStr

Synopsis: Checks whether a string contains a given substring.

Declaration: `function AnsiContainsStr(const AText: AnsiString;
 const ASubText: AnsiString) : Boolean
function AnsiContainsStr(const AText: UnicodeString;
 const ASubText: UnicodeString) : Boolean`

Visibility: default

Description: AnsiContainsString checks whether AText contains ASubText, and returns True if this is the case, or returns False otherwise. The search is performed case-sensitive.

Errors: None

See also: AnsiContainsText ([1300](#)), AnsiEndsStr ([1300](#)), AnsiIndexStr ([1301](#)), AnsiStartsStr ([1304](#))

74.3.4 AnsiContainsText

Synopsis: Check whether a string contains a certain substring, ignoring case.

Declaration:

```
function AnsiContainsText(const AText: AnsiString;
                        const ASubText: AnsiString) : Boolean
```

Visibility: default

Description: `AnsiContainsText` checks whether `AText` contains `ASubText`, and returns `True` if this is the case, or returns `False` otherwise. The search is performed case-insensitive.

See also: `AnsiContainsStr` ([1299](#)), `AnsiEndsText` ([1300](#)), `AnsiIndexText` ([1301](#)), `AnsiStartsText` ([1304](#))

74.3.5 AnsiEndsStr

Synopsis: Check whether a string ends with a certain substring.

Declaration:

```
function AnsiEndsStr(const ASubText: AnsiString;
                    const AText: AnsiString) : Boolean
function AnsiEndsStr(const ASubText: UnicodeString;
                    const AText: UnicodeString) : Boolean
```

Visibility: default

Description: `AnsiEndsStr` checks `AText` to see whether it ends with `ASubText`, and returns `True` if it does, `False` if not. The check is performed case-sensitive. Basically, it checks whether the position of `ASubText` equals the length of `AText` minus the length of `ASubText` plus one.

Errors: None.

See also: `AnsiEndsText` ([1300](#)), `AnsiStartsStr` ([1304](#)), `AnsiIndexStr` ([1301](#)), `AnsiContainsStr` ([1299](#))

74.3.6 AnsiEndsText

Synopsis: Check whether a string ends with a certain substring, ignoring case.

Declaration:

```
function AnsiEndsText(const ASubText: AnsiString;
                     const AText: AnsiString) : Boolean
function AnsiEndsText(const ASubText: UnicodeString;
                     const AText: UnicodeString) : Boolean
```

Visibility: default

Description: `AnsiEndsText` checks `AText` to see whether it ends with `ASubText`, and returns `True` if it does, `False` if not. The check is performed case-insensitive. Basically, it checks whether the position of `ASubText` equals the length of `AText` minus the length of `ASubText` plus one.

Errors: None

See also: `AnsiStartsText` ([1304](#)), `AnsiEndsStr` ([1300](#)), `AnsiIndexText` ([1301](#)), `AnsiContainsText` ([1300](#))

74.3.7 AnsiIndexStr

Synopsis: Searches, observing case, for a string in an array of strings.

Declaration: `function AnsiIndexStr(const AText: AnsiString;
const AValues: Array of AnsiString) : Integer`

Visibility: default

Description: `AnsiIndexStr` matches `AText` against each string in `AValues`. If a match is found, the corresponding index (zero-based) in the `AValues` array is returned. If no match is found, -1 is returned. The strings are matched observing case.

Errors: None.

See also: `AnsiIndexText` (1301), `AnsiMatchStr` (1302), `AnsiMatchText` (1302)

74.3.8 AnsiIndexText

Synopsis: Searches, case insensitive, for a string in an array of strings.

Declaration: `function AnsiIndexText(const AText: AnsiString;
const AValues: Array of AnsiString) : Integer`

Visibility: default

Description: `AnsiIndexText` matches `AText` against each string in `AValues`. If a match is found, the corresponding index (zero-based) in the `AValues` array is returned. If no match is found, -1 is returned. The strings are matched ignoring case.

Errors: None

See also: `AnsiIndexStr` (1301), `AnsiMatchStr` (1302), `IndexStr` (1315), `MatchStr` (1318), `AnsiMatchText` (1302)

74.3.9 AnsiLeftStr

Synopsis: Copies a number of characters starting at the left of a string.

Declaration: `function AnsiLeftStr(const AText: AnsiString; const ACount: SizeInt)
: AnsiString`

Visibility: default

Description: `AnsiLeftStr` returns the `ACount` leftmost characters from `AText`. If `ACount` is larger than the length of `AText`, only as much characters as available in `AText` will be copied. If `ACount` is zero or negative, no characters will be copied. The characters are counted as characters, not as Bytes. This function corresponds to the Visual Basic `LeftStr` function.

Errors: None.

See also: `AnsiMidStr` (1302), `AnsiRightStr` (1304), `LeftStr` (1317), `RightStr` (1324), `MidStr` (1318), `LeftBStr` (1317), `RightBStr` (1324), `MidBStr` (1318)

74.3.10 AnsiMatchStr

Synopsis: Check whether a string occurs in an array of strings, observing case.

Declaration: `function AnsiMatchStr(const AText: AnsiString;
const AValues: Array of AnsiString) : Boolean`

Visibility: default

Description: `AnsiMatchStr` matches `AText` against each string in `AValues`. If a match is found, it returns `True`, otherwise `False` is returned. The strings are matched observing case.

This function simply calls `AnsiIndexStr` (1301) and checks whether it returns -1 or not.

74.3.11 AnsiMatchText

Synopsis: Check whether a string occurs in an array of strings, disregarding case.

Declaration: `function AnsiMatchText(const AText: AnsiString;
const AValues: Array of AnsiString) : Boolean`

Visibility: default

Description: `AnsiIndexStr` matches `AText` against each string in `AValues`. If a match is found, it returns `True`, otherwise `False` is returned. The strings are matched ignoring case.

This function simply calls `AnsiIndexText` (1301) and checks whether it returns -1 or not.

74.3.12 AnsiMidStr

Synopsis: Returns a number of characters copied from a given location in a string.

Declaration: `function AnsiMidStr(const AText: AnsiString; const AStart: SizeInt;
const ACount: SizeInt) : AnsiString`

Visibility: default

Description: `AnsiMidStr` returns `ACount` characters from `AText`, starting at position `AStart`. If `AStart+ACount` is larger than the length of `AText`, only as much characters as available in `AText` (starting from `AStart`) will be copied. If `ACount` is zero or negative, no characters will be copied. The characters are counted as characters, not as Bytes.

This function corresponds to the Visual Basic `MidStr` function.

Errors: None

See also: `AnsiLeftStr` (1301), `AnsiRightStr` (1304), `LeftStr` (1317), `RightStr` (1324), `MidStr` (1318), `LeftBStr` (1317), `RightBStr` (1324), `MidBStr` (1318)

74.3.13 AnsiProperCase

Synopsis: Pretty-Print a string: make lowercase and capitalize first letters of words.

Declaration: `function AnsiProperCase(const S: string; const WordDelims: TSysCharSet)
: string`

Visibility: default

Description: `AnsiProperCase` converts `S` to an all lowercase string, but capitalizes the first letter of every word in the string, and returns the resulting string. When searching for words, the characters in `WordDelimiters` are used to determine the boundaries of words. The constant `StdWordDelims` (1297) can be used for this.

74.3.14 AnsiReplaceStr

Synopsis: Search and replace all occurrences of a string, case sensitive.

Declaration:

```
function AnsiReplaceStr(const AText: AnsiString;
                        const AFromText: AnsiString;
                        const AToText: AnsiString) : AnsiString
```

Visibility: default

Description: `AnsiReplaceString` searches `AText` for all occurrences of the string `AFromText` and replaces them with `AToText`, and returns the resulting string. The search is performed observing case.

Errors: None.

See also: `AnsiReplaceText` ([1303](#)), `SearchBuf` ([1326](#))

74.3.15 AnsiReplaceText

Synopsis: Search and replace all occurrences of a string, case insensitive.

Declaration:

```
function AnsiReplaceText(const AText: AnsiString;
                        const AFromText: AnsiString;
                        const AToText: AnsiString) : AnsiString
```

Visibility: default

Description: `AnsiReplaceString` searches `AText` for all occurrences of the string `AFromText` and replaces them with `AToText` , and returns the resulting string. The search is performed ignoring case.

Errors: None.

See also: `AnsiReplaceStr` ([1303](#)), `SearchBuf` ([1326](#))

74.3.16 AnsiResemblesText

Synopsis: Check whether 2 strings resemble each other.

Declaration:

```
function AnsiResemblesText(const AText: AnsiString;
                          const AOther: AnsiString) : Boolean
```

Visibility: default

Description: `AnsiResemblesText` will check whether `AnsiResemblesProc` ([1296](#)) is set. If it is not set, `False` is returned. If it is set, `AText` and `AOtherText` are passed to it and it's result is returned.

Errors: None.

See also: `AnsiResemblesProc` ([1296](#)), `SoundexProc` ([1328](#))

74.3.17 AnsiReverseString

Synopsis: Reverse the letters in a string.

Declaration: `function AnsiReverseString(const AText: AnsiString) : AnsiString`

Visibility: default

Description: `AnsiReverseString` returns a string with all characters of `AText` in reverse order.
if the result of this function equals `AText`, `AText` is called a palindrome.

Errors: None.

74.3.18 AnsiRightStr

Synopsis: Copies a number of characters starting at the right of a string.

Declaration: `function AnsiRightStr(const AText: AnsiString; const ACount: SizeInt)
: AnsiString`

Visibility: default

Description: `AnsiLeftStr` returns the `ACount` rightmost characters from `AText`. If `ACount` is larger than the length of `AText`, only as much characters as available in `AText` will be copied. If `ACount` is zero or negative, no characters will be copied. The characters are counted as characters, not as Bytes.
This function corresponds to the Visual Basic `RightStr` function.

Errors: None.

See also: `AnsiLeftStr` (1301), `AnsiMidStr` (1302), `LeftStr` (1317), `RightStr` (1324), `MidStr` (1318), `LeftBStr` (1317), `RightBStr` (1324), `MidBStr` (1318)

74.3.19 AnsiStartsStr

Synopsis: Check whether a string starts with a given substring, observing case.

Declaration: `function AnsiStartsStr(const ASubText: AnsiString;
const AText: AnsiString) : Boolean
function AnsiStartsStr(const ASubText: UnicodeString;
const AText: UnicodeString) : Boolean`

Visibility: default

Description: `AnsiStartsStr` checks `AText` to see whether it starts with `ASubText`, and returns `True` if it does, `False` if not. The check is performed case-sensitive. Basically, it checks whether the position of `ASubText` equals 1.

See also: `AnsiEndsStr` (1300), `AnsiStartsStr` (1304), `AnsiIndexStr` (1301), `AnsiContainsStr` (1299)

74.3.20 AnsiStartsText

Synopsis: Check whether a string starts with a given substring, ignoring case.

Declaration: `function AnsiStartsText(const ASubText: AnsiString;
const AText: AnsiString) : Boolean`

Visibility: default

Description: `AnsiStartsText` checks `AText` to see whether it starts with `ASubText`, and returns `True` if it does, `False` if not. The check is performed case-insensitive. Basically, it checks whether the position of `ASubText` equals 1.

Errors: None.

See also: `AnsiEndsText` ([1300](#)), `AnsiStartsStr` ([1304](#)), `AnsiIndexText` ([1301](#)), `AnsiContainsText` ([1300](#))

74.3.21 BinToHex

Synopsis: Convert a binary buffer to a hexadecimal string.

Declaration:

```

procedure BinToHex(const BinBuffer: TBytes; BinBufOffset: Integer;
                   var HexBuffer: TBytes; HexBufOffset: Integer;
                   Count: Integer); Overload
procedure BinToHex(BinValue: Pointer; HexValue: PWideChar;
                   BinBufSize: Integer); Overload
procedure BinToHex(const BinValue; HexValue: PWideChar;
                   BinBufSize: Integer); Overload
procedure BinToHex(BinValue: PAnsiChar; HexValue: PAnsiChar;
                   BinBufSize: Integer); Overload
procedure BinToHex(BinValue: PAnsiChar; HexValue: PWideChar;
                   BinBufSize: Integer); Overload
procedure BinToHex(const BinValue; HexValue: PAnsiChar;
                   BinBufSize: Integer); Overload
procedure BinToHex(BinValue: Pointer; HexValue: PAnsiChar;
                   BinBufSize: Integer); Overload

```

Visibility: default

Description: `BinToHex` converts the byte values in `BinValue` to a string consisting of 2-character hexadecimal strings in `HexValue`. `BufSize` specifies the length of `BinValue`, which means that `HexValue` must have size $2 * \text{BufSize}$.

For example a buffer containing the byte values 255 and 0 will be converted to FF00.

Errors: No length checking is done, so if an invalid size is specified, an exception may follow.

See also: `HexToBin` ([1313](#))

74.3.22 ContainsStr

Synopsis: Check whether one text contains another (case sensitive).

Declaration:

```

function ContainsStr(const AText: string; const ASubText: string)
                   : Boolean

```

Visibility: default

Description: `ContainsStr` is an alias for `AnsiContainsStr` ([1299](#))

See also: `AnsiContainsStr` ([1299](#))

74.3.23 ContainsText

Synopsis: Check whether one text contains another (case insensitive).

Declaration: `function ContainsText(const AText: string; const ASubText: string) : Boolean`

Visibility: default

Description: `ContainsText` is an alias for `AnsiContainsText` (1300)

See also: `AnsiContainsText` (1300)

74.3.24 Copy2Space

Synopsis: Returns all characters in a string till the first space character (not included).

Declaration: `function Copy2Space(const S: string) : string`

Visibility: default

Description: `Copy2Space` determines the position of the first space in the string `S` and returns all characters up to this position. The space character itself is not included in the result string. The string `S` is left untouched. If there is no space in `S`, then the whole string `S` is returned.

This function simply calls `Copy2Symb` (1306) with the space (ASCII code 32) as the symbol argument.

Errors: None.

See also: `Copy2Symb` (1306), `Copy2SpaceDel` (1306)

74.3.25 Copy2SpaceDel

Synopsis: Deletes and returns all characters in a string till the first space character (not included).

Declaration: `function Copy2SpaceDel(var S: string) : string`

Visibility: default

Description: `Copy2SpaceDel` determines the position of the first space in the string `S` and returns all characters up to this position. The space character itself is not included in the result string. All returned characters, including the space, are deleted from the string `S`, after which it is right-trimmed. If there is no space in `S`, then the whole string `S` is returned, and `S` itself is emptied.

This function simply calls `Copy2SymbDel` (1307) with the space (ASCII code 32) as the symbol argument.

Errors: None.

See also: `Copy2SymbDel` (1307), `Copy2Space` (1306)

74.3.26 Copy2Symb

Synopsis: Returns all characters in a string till a given character (not included).

Declaration: `function Copy2Symb(const S: string; Symb: AnsiChar) : string`

Visibility: default

Description: `Copy2Symb` determines the position of the first occurrence of `Symb` in the string `S` and returns all characters up to this position. The `Symb` character itself is not included in the result string. The string `S` is left untouched. If `Symb` does not appear in `S`, then the whole of `S` is returned.

Errors: None.

See also: `Copy2Space` ([1306](#)), `Copy2SymbDel` ([1307](#))

74.3.27 Copy2SymbDel

Synopsis: Deletes and returns all characters in a string till a given character (not included).

Declaration: `function Copy2SymbDel (var S: string; Symb: AnsiChar) : string`

Visibility: default

Description: `Copy2SymbDel` determines the position of the first occurrence of `Symb` in the string `S` and returns all characters up to this position. The `Symb` character itself is not included in the result string. All returned characters and the `Symb` character, are deleted from the string `S`, after which it is right-trimmed. If `Symb` does not appear in `S`, then the whole of `S` is returned, and `S` itself is emptied.

Errors: None.

See also: `Copy2SpaceDel` ([1306](#)), `Copy2Symb` ([1306](#))

74.3.28 Dec2Numb

Synopsis: Convert a decimal number to a string representation, using given a base.

Declaration: `function Dec2Numb (N: LongInt; Len: Byte; Base: Byte) : string`

Visibility: default

Description: `Dec2Numb` converts `N` to its representation using base `Base`. `N` must be a positive integer. The resulting string is left-padded with zeroes till it has length `Len`. `Base` must be in the range 2-36 to be meaningful, but no checking on this is performed.

Errors: If `Base` is out of range, the resulting string will contain unreadable (non-alphanumeric) characters.

See also: `Hex2Dec` ([1313](#)), `IntToBin` ([1315](#)), `intToRoman` ([1316](#)), `RomanToInt` ([1325](#))

74.3.29 DecodeSoundexInt

Synopsis: Decodes the integer representation of a soundex code and returns the original soundex code.

Declaration: `function DecodeSoundexInt (AValue: Integer) : string`

Visibility: default

Description: `DecodeSoundexInt` converts the integer value `AValue` to a soundex string. It performs the reverse operation of the `SoundexInt` ([1327](#)) function. The result is the soundex string corresponding to `AValue`.

Errors: None.

See also: `SoundexInt` ([1327](#)), `DecodeSoundexWord` ([1308](#)), `Soundex` ([1327](#))

74.3.30 DecodeSoundexWord

Synopsis: Decodes the word-sized representation of a soundex code and returns the original soundex code.

Declaration: `function DecodeSoundexWord(AValue: Word) : string`

Visibility: default

Description: `DecodeSoundexWord` converts the integer value `AValue` to a soundex string. It performs the reverse operation of the `SoundexWord` (1328) function. The result is the soundex string corresponding to `AValue`.

Errors: None.

See also: `SoundexInt` (1327), `DecodeSoundexInt` (1307), `Soundex` (1327)

74.3.31 DelChars

Synopsis: Delete all occurrences of a given character from a string.

Declaration: `function DelChars(const S: string; Chr: AnsiChar) : string`
`function DelChars(const S: string; Chars: TSysCharSet) : string`

Visibility: default

Description: `DelChars` returns a copy of `S` with all `Chr` characters removed from it.

Errors: None.

See also: `DelSpace` (1308), `DelSpace1` (1308)

74.3.32 DelSpace

Synopsis: Delete all occurrences of a space from a string.

Declaration: `function DelSpace(const S: string) : string`

Visibility: default

Description: `DelSpace` returns a copy of `S` with all spaces (ASCII code 32) removed from it.

Errors: None.

See also: `DelChars` (1308), `DelSpace1` (1308)

74.3.33 DelSpace1

Synopsis: Reduces sequences of space characters to 1 space character.

Declaration: `function DelSpace1(const S: string) : string`

Visibility: default

Description: `DelSpace1` returns a copy of `S` with all sequences of spaces reduced to 1 space.

Errors: None.

See also: `DelChars` (1308), `DelSpace` (1308)

74.3.34 DupeString

Synopsis: Creates and concatenates N copies of a string.

Declaration: `function DupeString(const AText: string; ACount: Integer) : string`

Visibility: default

Description: `DupeString` returns a string consisting of `ACount` concatenations of `AText`. Thus

```
DupeString('1234567890', 3);
```

will produce a string

```
'123456789012345678901234567890'
```

If `aCount` ≤ 0 then the result is an empty string.

Errors: If you specify a too large `aCount` (so the resulting string would require more memory than allowed on your system) then an `EOutOfMemory` exception can be raised.

74.3.35 EndsStr

Synopsis: Check whether one string ends with another.

Declaration: `function EndsStr(const ASubText: string; const AText: string) : Boolean`

Visibility: default

Description: `EndsStr` checks whether `aText` ends with `aSubText` and returns `True` if it does. i.e. it returns `true` if the last characters of `aText` are `aSubText`. It follows that the length of `aText` must be at least the length of `aSubText`. The comparison is made case-sensitive. If you wish to compare case-insensitively, use `EndsText` (1309) instead.

See also: `AnsiEndsStr` (1300), `EndsText` (1309), `StartsText` (1330)

74.3.36 EndsText

Synopsis: Check whether one text ends with another.

Declaration: `function EndsText(const ASubText: string; const AText: string) : Boolean`

Visibility: default

Description: `EndsText` checks whether `aText` ends with `aSubText` and returns `True` if it does. i.e. it returns `true` if the last characters of `aText` are `aSubText`. It follows that the length of `aText` must be at least the length of `aSubText`. The comparison is made case-insensitive. If you wish to compare case-sensitively, use `EndsStr` (1309) instead.

See also: `AnsiEndsStr` (1300), `EndsStr` (1309), `StartsText` (1330)

74.3.37 ExtractDelimited

Synopsis: Extract the N-th delimited part from a string.

Declaration: `function ExtractDelimited(N: Integer; const S: string;
const Delims: TSysCharSet) : string`

Visibility: default

Description: `ExtractDelimited` extracts the N-th part from the string `S`. The set of characters in `Delims` are used to mark part boundaries. When a delimiter is encountered, a new part is started and the old part is ended. Another way of stating this is that any (possibly empty) series of characters not in `Delims`, situated between 2 characters in `Delims`, it is considered as piece of a part. This means that if 2 delimiter characters appear next to each other, there is an empty part between it. If an N-th part cannot be found, an empty string is returned. However, unlike `ExtractWord` (1310), an empty string is a valid return value, i.e. a part can be empty.

The predefined constant `StdWordDelims` (1297) can be used for the `Delims` argument. The predefined constant `Brackets` (1296) would be better suited the `Delims` argument e.g. in case factors in a mathematical expression are searched.

Errors: None.

See also: `ExtractSubStr` (1310), `ExtractWord` (1310), `ExtractWordPos` (1311)

74.3.38 ExtractSubstr

Synopsis: Extract a word from a string, starting at a given position in the string.

Declaration: `function ExtractSubstr(const S: string; var Pos: Integer;
const Delims: TSysCharSet) : string`

Visibility: default

Description: `ExtractSubStr` returns all characters from `S` starting at position `Pos` till the first character in `Delims`, or till the end of `S` is reached. The delimiter character is not included in the result. `Pos` is then updated to point to the next first non-delimiter character in `S`. If `Pos` is larger than the `Length` of `S`, an empty string is returned.

The predefined constant `StdWordDelims` (1297) can be used for the `Delims` argument.

Errors: None.

See also: `ExtractDelimited` (1310), `ExtractWord` (1310), `ExtractWordPos` (1311)

74.3.39 ExtractWord

Synopsis: Extract the N-th word out of a string.

Declaration: `function ExtractWord(N: Integer; const S: string;
const WordDelims: TSysCharSet) : string`

Visibility: default

Description: `ExtractWord` extracts the N-th word from the string `S`. The set of characters in `WordDelims` are used to mark word boundaries. A word is defined as any non-empty sequence of characters which are not present in `WordDelims`: if a character is not in `WordDelims`, it is considered as part of a word. If an N-th word cannot be found, an empty string is returned.

Unlike `ExtractDelimited` (1310), an empty string is not a valid return value, i.e. is not a word. If an empty string is returned, the index `N` was out of range.

The predefined constant `StdWordDelims` (1297) can be used for the `WordDelims` argument.

Errors: None.

See also: `ExtractWordPos` (1311), `ExtractSubStr` (1310), `ExtractDelimited` (1310), `IsWordPresent` (1317), `WordCount` (1332), `WordPosition` (1333)

74.3.40 ExtractWordPos

Synopsis: Extract a word from a string, and return the position where it was located in the string.

Declaration:

```
function ExtractWordPos(N: Integer; const S: string;
                        const WordDelims: TSysCharSet; out Pos: Integer)
                        : string
```

Visibility: default

Description: `ExtractWordPos` extracts the `N`-th word from the string `S` and returns the position of this word in `Pos`. The set of characters in `WordDelims` are used to mark word boundaries. A word is defined as any non-empty sequence of characters which are not present in `WordDelims`: if a character is not in `WordDelims`, it is considered as part of a word. If an `N`-th word cannot be found, an empty string is returned and `Pos` is zero.

Unlike `ExtractDelimited` (1310), an empty string is not a valid return value, i.e. is not a word. If an empty string is returned, the index `N` was out of range.

The predefined constant `StdWordDelims` (1297) can be used for the `WordDelims` argument.

Errors: None.

See also: `ExtractWord` (1310), `ExtractSubStr` (1310), `IsWordPresent` (1317), `WordCount` (1332), `WordPosition` (1333)

74.3.41 FindMatchesBoyerMooreCaseInsensitive

Synopsis: Find case-insensitive matches of a string using a Boyer-Moore algorithm.

Declaration:

```
function FindMatchesBoyerMooreCaseInsensitive(const S: PAnsiChar;
                                              const OldPattern: PAnsiChar;
                                              const SSize: SizeInt;
                                              const OldPatternSize: SizeInt;
                                              out aMatches: SizeIntArray;
                                              const aMatchAll: Boolean)
                                              : Boolean

function FindMatchesBoyerMooreCaseInsensitive(const S: string;
                                              const OldPattern: string;
                                              out aMatches: SizeIntArray;
                                              const aMatchAll: Boolean)
                                              : Boolean
```

Visibility: default

Description: `FindMatchesBoyerMooreCaseInsensitive` finds occurrences of `OldPattern` (with length `OldPatternSize`) in `S` (with length `SSize`). The search is performed case-insensitively, and all (zero based) positions are reported in `aMatches`. If `aMatchAll` is `True`, all positions will be reported. If `aMatchAll` is `False`, only the first position is reported.

Errors: None.

See also: [FindMatchesBoyerMooreCaseSensitive \(1312\)](#), [StringReplace \(1330\)](#)

74.3.42 FindMatchesBoyerMooreCaseSensitive

Synopsis: Find case-sensitive matches of a string using a Boyer-Moore algorithm.

Declaration:

```
function FindMatchesBoyerMooreCaseSensitive(const S: PAnsiChar;
                                           const OldPattern: PAnsiChar;
                                           const SSize: SizeInt;
                                           const OldPatternSize: SizeInt;
                                           out aMatches: SizeIntArray;
                                           const aMatchAll: Boolean)
                                           : Boolean

function FindMatchesBoyerMooreCaseSensitive(const S: string;
                                           const OldPattern: string;
                                           out aMatches: SizeIntArray;
                                           const aMatchAll: Boolean)
                                           : Boolean
```

Visibility: default

Description: `FindMatchesBoyerMooreCaseSensitive` finds occurrences of `OldPattern` (with length `OldPatternSize`) in `S` (with length `SSize`). The search is performed case-sensitively, and all (zero based) positions are reported in `aMatches`. If `aMatchAll` is `True`, all positions will be reported. If `aMatchAll` is `False`, only the first position is reported.

Errors: None.

See also: [FindMatchesBoyerMooreCaseInSensitive \(1311\)](#), [StringReplace \(1330\)](#)

74.3.43 FindPart

Synopsis: Search for a substring in a string, using wildcards.

Declaration:

```
function FindPart(const HelpWilds: string; const InputStr: string)
                  : SizeInt
```

Visibility: default

Description: `FindPart` searches the string `InputStr` and returns the first string that matches the wildcards specification in `HelpWilds`. If no match is found, an empty string is returned. Currently, the only valid wildcards is the "?" character.

Errors: None.

See also: [SearchBuf \(1326\)](#)

74.3.44 GetCmdLineArg

Synopsis: Returns the command-line argument following the given switch.

Declaration:

```
function GetCmdLineArg(const Switch: string; SwitchChars: TSysCharSet)
                      : string
```

Visibility: default

Description: `GetCmdLineArg` returns the value for the `Switch` option on the command-line, if any is given. Command-line arguments are considered switches if they start with one of the characters in the `SwitchChars` set. The value is the command-line argument following the switch command-line argument.

Gnu-style (long) Options of the form `switch=value` are not supported.

The `StdSwitchChars` (1297) constant can be used as value for the `SwitchChars` parameter.

Errors: The `GetCmdLineArg` does not check whether the value of the option does not start with a switch character. i.e.

```
myprogram -option1 -option2
```

will result in "-option2" as the result of the `GetCmdLineArg` call for `option1`.

See also: `StdSwitchChars` (1297)

74.3.45 Hex2Dec

Synopsis: Converts a hexadecimal string to a decimal value.

Declaration: `function Hex2Dec(const S: string) : LongInt`

Visibility: default

Description: `Hex2Dec` converts the hexadecimal value in the string `S` to its decimal value. Unlike the standard `Val` or `StrToInt` functions, there need not be a \$ sign in front of the hexadecimal value to indicate that it is indeed a hexadecimal value.

Errors: If `S` does not contain a valid hexadecimal value, an `EConvertError` exception will be raised.

See also: `Dec2Numb` (1307), `IntToBin` (1315), `intToRoman` (1316), `RomanToInt` (1325)

74.3.46 Hex2Dec64

Synopsis: Convert hexadecimal value to 64-bit integer.

Declaration: `function Hex2Dec64(const S: string) : Int64`

Visibility: default

Description: `Hex2Dec64` converts the hexadecimal number in `S` to a 64-bit value and returns the result. It is equivalent to `StrToInt('$'+S)`.

See also: `Hex2Dec` (1313)

74.3.47 HexToBin

Synopsis: Convert a hexadecimal string to a binary buffer.

Declaration: `function HexToBin(HexText: PAnsiChar; BinBuffer: PAnsiChar;
 BinBufSize: Integer) : Integer; Overload
function HexToBin(const HexText: PWideChar; HexTextOffset: Integer;
 var BinBuffer: TBytes; BinBufOffset: Integer;
 Count: Integer) : Integer; Overload
function HexToBin(const HexText: TBytes; HexTextOffset: Integer;`

```

        var BinBuffer: TBytes; BinBufOffset: Integer;
        Count: Integer) : Integer; Overload
function HexToBin(HexText: PWideChar; BinBuffer: Pointer;
        BinBufSize: Integer) : Integer; Overload
function HexToBin(const HexText: PWideChar; var BinBuffer;
        BinBufSize: Integer) : Integer; Overload
function HexToBin(HexText: PWideChar; BinBuffer: PAnsiChar;
        BinBufSize: Integer) : Integer; Overload
function HexToBin(HexText: PAnsiChar; var BinBuffer;
        BinBufSize: Integer) : Integer; Overload
function HexToBin(const HexText: PAnsiChar; BinBuffer: Pointer;
        BinBufSize: Integer) : Integer; Overload

```

Visibility: default

Description: HexToBin scans the hexadecimal string representation in HexValue and transforms every 2 character hexadecimal number to a byte and stores it in BinValue. The buffer size is the size of the binary buffer. Scanning will stop if the size of the binary buffer is reached or when an invalid character is encountered. The return value is the number of stored bytes.

Errors: No length checking is done, so if an invalid size is specified, an exception may follow.

See also: BinToHex ([1305](#))

74.3.48 IfThen

Synopsis: Returns one of two strings, depending on a boolean expression.

Declaration:

```

function IfThen(AValue: Boolean; const ATrue: string;
        const AFalse: string) : string; Overload
function IfThen(AValue: Boolean; const ATrue: TStringDynArray;
        const AFalse: TStringDynArray) : TStringDynArray
; Overload

```

Visibility: default

Description: IfThen returns ATrue if AValue is True, and returns AFalse if AValue is false.

Errors: None.

See also: AnsiMatchStr ([1302](#)), AnsiMatchText ([1302](#))

74.3.49 in(Ansistring,):Boolean

Declaration:

```

operator in(const AText: Ansistring; const AValues: Array of Ansistring)
: Boolean

```

Visibility: default

74.3.50 in(UnicodeString,):Boolean

Synopsis: Check presence of string in an array of strings.

Declaration:

```

operator in(const AText: UnicodeString;
        const AValues: Array of UnicodeString) : Boolean

```

Visibility: default

Description: This is the (unicode string) operator equivalent of `IndexStr` ([1315](#))

See also: `IndexStr` ([1315](#))

74.3.51 IndexStr

Synopsis: Searches, observing case, for a string in an array of strings.

Declaration:

```
function IndexStr(const AText: UnicodeString;
                  const AValues: Array of UnicodeString) : Integer
function IndexStr(const AText: Ansistring;
                  const AValues: Array of Ansistring) : Integer
```

Visibility: default

Description: `IndexStr` matches `AText` against each Unicode string in `AValues`. If a match is found, the corresponding index (zero-based) in the `AValues` array is returned. If no match is found, -1 is returned. The strings are matched observing case.

Errors: None.

See also: `AnsiIndexText` ([1301](#)), `MatchStr` ([1318](#)), `AnsiIndexStr` ([1301](#))

74.3.52 IndexText

Synopsis: Index of text in a list of values.

Declaration:

```
function IndexText(const AText: Ansistring;
                   const AValues: Array of Ansistring) : Integer
function IndexText(const AText: UnicodeString;
                   const AValues: Array of UnicodeString) : Integer
```

Visibility: default

Description: `IndexText` returns the index of the string in the array `aValues` that matches `aText`, and -1 otherwise. The comparison is done case insensitively. If you wish to compare case sensitively, use `IndexStr` ([1315](#)) instead.

See also: `IndexStr` ([1315](#)), `MatchText` ([1318](#)), `AnsiIndexText` ([1301](#))

74.3.53 IntToBin

Synopsis: Converts an integer to a binary string representation, inserting spaces at fixed locations.

Declaration:

```
function IntToBin(Value: LongInt; Digits: Integer; Spaces: Integer)
                  : string
function IntToBin(Value: LongInt; Digits: Integer) : string
function IntToBin(Value: Int64; Digits: Integer) : string
```

Visibility: default

Description: `IntToBin` converts `Value` to a string with it's binary (base 2) representation. The resulting string contains `Digits` digits, with spaces inserted every `Spaces` digits. `Spaces` equal to zero yields a result without spacing. If `Digits` is larger than 32, it is truncated to 32.

See also: `Hex2Dec` ([1313](#)), `IntToRoman` ([1316](#))

74.3.54 IntToRoman

Synopsis: Represent an integer with roman numerals.

Declaration: `function IntToRoman(Value: LongInt) : string`

Visibility: default

Description: `IntToRoman` converts `Value` to a string with the Roman representation of `Value`. Number up to 1 million can be represented this way.

Errors: None.

See also: `RomanToInt` ([1325](#)), `Hex2Dec` ([1313](#)), `IntToBin` ([1315](#))

74.3.55 IsEmptyStr

Synopsis: Check whether a string is empty, disregarding whitespace characters.

Declaration: `function IsEmptyStr(const S: string; const EmptyChars: TSysCharSet) : Boolean`

Visibility: default

Description: `IsEmptyStr` returns `True` if the string `S` only contains characters whitespace characters, all characters in `EmptyChars` are considered whitespace characters. If a character not present in `EmptyChars` is found in `S`, `False` is returned.

Errors: None.

See also: `IsWild` ([1316](#)), `FindPart` ([1312](#)), `IsWordPresent` ([1317](#))

74.3.56 IsWild

Synopsis: Check whether a string matches a wildcard search expression.

Declaration: `function IsWild(InputStr: string; Wilds: string; IgnoreCase: Boolean) : Boolean`

Visibility: default

Description: `IsWild` checks `InputStr` for the presence of the `Wilds` string. `Wilds` may contain "?" and "*" wildcard characters, which have their usual meaning: "*" matches any series of characters, possibly empty. "?" matches any single character. The function returns `True` if a string is found that matches `Wilds`, `False` otherwise.

If `IgnoreCase` is `True`, the non-wildcard characters are matched case insensitively. If it is `False`, case is observed when searching.

Errors: None.

See also: `SearchBuf` ([1326](#)), `FindPart` ([1312](#))

74.3.57 IsWordPresent

Synopsis: Check for the presence of a word in a string.

Declaration: `function IsWordPresent(const W: string; const S: string;
const WordDelims: TSysCharSet) : Boolean`

Visibility: default

Description: `IsWordPresent` checks for the presence of the word `W` in the string `S`. Words are delimited by the characters found in `WordDelims`. The function returns `True` if a match is found, `False` otherwise. The search is performed case sensitive.

This function is equivalent to the `SearchBuf` (1326) function with the `soWholeWords` option specified.

Errors: None.

See also: `SearchBuf` (1326)

74.3.58 LeftBStr

Synopsis: Copies Count characters starting at the left of a string.

Declaration: `function LeftBStr(const AText: AnsiString; const AByteCount: SizeInt)
: AnsiString`

Visibility: default

Description: `LeftBStr` returns a string containing the leftmost `AByteCount` bytes from the string `AText`. If `AByteCount` is larger than the length (in bytes) of `AText`, only as many bytes as available are returned.

Errors: None.

See also: `LeftStr` (1317), `AnsiLeftStr` (1301), `RightBStr` (1324), `MidBStr` (1318)

74.3.59 LeftStr

Synopsis: Copies Count characters starting at the left of a string.

Declaration: `function LeftStr(const AText: AnsiString; const ACount: SizeInt)
: AnsiString
function LeftStr(const AText: WideString; const ACount: SizeInt)
: WideString`

Visibility: default

Description: `LeftStr` returns a string containing the leftmost `ACount` characters from the string `AText`. If `ACount` is larger than the length (in characters) of `AText`, only as many characters as available are returned.

Errors: None.

See also: `LeftBStr` (1317), `AnsiLeftStr` (1301), `RightStr` (1324), `MidStr` (1318)

74.3.60 MatchStr

Synopsis: Check whether a string occurs in an array of strings, observing case.

Declaration:

```
function MatchStr(const AText: UnicodeString;
                  const AValues: Array of UnicodeString) : Boolean
function MatchStr(const AText: Ansistring;
                  const AValues: Array of Ansistring) : Boolean
```

Visibility: default

Description: `MatchStr` matches `AText` against each Unicode string in `AValues`. If a match is found, it returns `True`, otherwise `False` is returned. The strings are matched observing case.

This function simply calls `IndexStr` (1315) and checks whether it returns -1 or not.

74.3.61 MatchText

Synopsis: Check if a string is in a list of values.

Declaration:

```
function MatchText(const AText: Ansistring;
                   const AValues: Array of Ansistring) : Boolean
function MatchText(const AText: UnicodeString;
                   const AValues: Array of UnicodeString) : Boolean
```

Visibility: default

Description: `MatchText` returns `True` if `aText` equals one of the strings in `aValues`. The comparison is done case insensitively. If you wish to compare case sensitively, use `MatchStr` (1318) instead.

See also: `MatchStr` (1318), `AnsiMatchText` (1302), `IndexText` (1315)

74.3.62 MidBStr

Synopsis: Copies a number of characters starting at a given position in a string.

Declaration:

```
function MidBStr(const AText: AnsiString; const AByteStart: SizeInt;
                 const AByteCount: SizeInt) : AnsiString
```

Visibility: default

Description: `MidBStr` returns a string containing the first `AByteCount` bytes from the string `AText` starting at position `AByteStart`. If `AByteStart+AByteCount` is larger than the length (in bytes) of `AText`, only as many bytes as available are returned. If `AByteStart` is less than 1 or larger than the length of `AText`, then no characters are returned.

Errors: None.

See also: `LeftBStr` (1317), `AnsiMidStr` (1302), `RightBStr` (1324), `MidStr` (1318)

74.3.63 MidStr

Synopsis: Copies a number of characters starting at a given position in a string.

Declaration:

```
function MidStr(const AText: AnsiString; const AStart: SizeInt;
                const ACount: SizeInt) : AnsiString
function MidStr(const AText: WideString; const AStart: SizeInt;
                const ACount: SizeInt) : WideString
```

Visibility: default

Description: `MidStr` returns a string containing the first `ACount` bytes from the string `AText` starting at position `AStart`. If `AStart+ACount` is larger than the length (in characters) of `AText`, only as many characters as available are returned. If `AStart` is less than 1 or larger than the length of `AText`, then no characters are returned.

This function is equivalent to the standard `Copy` function, and is provided for completeness only.

Errors: None.

See also: `LeftStr` ([1317](#)), `AnsiMidStr` ([1302](#)), `RightStr` ([1324](#)), `MidBStr` ([1318](#))

74.3.64 NaturalCompareText

Synopsis: Compare using natural sort.

Declaration:

```
function NaturalCompareText(const S1: string; const S2: string)
    : Integer
function NaturalCompareText(const Str1: string; const Str2: string;
    const ADecSeparator: AnsiChar;
    const AThousandSeparator: AnsiChar) : Integer
```

Visibility: default

Description: `NaturalCompareText` will compare 2 strings and return one of the following values:

- 1 If `S1` comes before `S2`
- 0 If `S1` equals `S2`
- 1 If `S1` comes after `S2`

When `S1` and `S2` are integer or floating point values, the actual values are compared. Thus '12' will come after '2'. If either of the values is not a valid integer or floating point values, the strings are compared case insensitively as texts using `CompareText` ([1669](#))

A decimal separator and thousands separator can optionally be specified in `ADecSeparator`, `AThousandSeparator`. If none are specified, the defaults from the system unit will be used.

See also: `CompareText` ([1296](#))

74.3.65 NPos

Synopsis: Returns the position of the N-th occurrence of a substring in a string.

Declaration:

```
function NPos(const C: string; S: string; N: Integer) : SizeInt
```

Visibility: default

Description: `NPos` checks `S` for the position of the N-th occurrence of `C`. If `C` occurs less than `N` times in `S`, or does not occur in `S` at all, 0 is returned. If `N` is less than 1, zero is returned.

Errors: None.

See also: `WordPosition` ([1333](#)), `FindPart` ([1312](#)), `#rtl.system.Pos` ([1515](#)), `PosEx` ([1321](#))

74.3.66 Numb2Dec

Synopsis: Converts a string representation of a number to its numerical value, given a certain base.

Declaration: `function Numb2Dec(S: string; Base: Byte) : LongInt`

Visibility: default

Description: `Numb2Dec` converts the number in string `S` to a decimal value. It assumes the number is represented using `Base` as the base. No checking is performed to see whether `S` contains a valid number using base `Base`.

Errors: None.

See also: `Hex2Dec` ([1313](#)), `Numb2USA` ([1320](#))

74.3.67 Numb2USA

Synopsis: Insert thousand separators.

Declaration: `function Numb2USA(const S: string) : string`

Visibility: default

Description: `Numb2USA` inserts thousand separators in the string `S` at the places where they are supposed to be, i.e. every 3 digits. The string `S` should contain a valid integer number, i.e. no digital number. No checking on this is done.

Errors: None.

74.3.68 PadCenter

Synopsis: Pad the string to a certain length, so the string is centered.

Declaration: `function PadCenter(const S: string; Len: SizeInt) : string`

Visibility: default

Description: `PadCenter` add spaces to the left and right of the string `S` till the result reaches length `Len`. If the number of spaces to add is odd, then the extra space will be added at the end. If the string `S` has length equal to or larger than `Len`, no spaces are added, and the string `S` is returned as-is.

Errors: None.

See also: `PadLeft` ([1320](#)), `PadRight` ([1321](#)), `AddChar` ([1299](#)), `AddCharR` ([1299](#))

74.3.69 PadLeft

Synopsis: Add spaces to the left of a string till a certain length is reached.

Declaration: `function PadLeft(const S: string; N: Integer) : string`

Visibility: default

Description: `PadLeft` add spaces to the left of the string `S` till the result reaches length `Len`. If the string `S` has length equal to or larger than `Len`, no spaces are added, and the string `S` is returned as-is. The resulting string is `S`, right-justified on length `Len`.

Errors: None.

See also: `PadLeft` ([1320](#)), `PadCenter` ([1320](#)), `AddChar` ([1299](#)), `AddCharR` ([1299](#))

74.3.70 PadRight

Synopsis: Add spaces to the right of a string till a certain length is reached.

Declaration: `function PadRight(const S: string; N: Integer) : string`

Visibility: default

Description: `PadRight` add spaces to the right of the string `S` till the result reaches length `Len`. If the string `S` has length equal to or larger than `Len`, no spaces are added, and the string `S` is returned as-is. The resulting string is `S`, left-justified on length `Len`.

Errors: None.

See also: `PadLeft` ([1320](#)), `PadCenter` ([1320](#)), `AddChar` ([1299](#)), `AddCharR` ([1299](#))

74.3.71 PosEx

Synopsis: Search for the occurrence of a character in a string, starting at a certain position.

Declaration: `function PosEx(const SubStr: Ansistring; const S: Ansistring;
 Offset: SizeUInt) : SizeInt`
`function PosEx(const SubStr: Ansistring; const S: Ansistring) : SizeInt`
`function PosEx(c: AnsiChar; const S: AnsiString; Offset: SizeUInt)
 : SizeInt`
`function PosEx(const SubStr: UnicodeString; const S: UnicodeString;
 Offset: SizeUInt) : SizeInt`
`function PosEx(c: WideChar; const S: UnicodeString; Offset: SizeUInt)
 : SizeInt`
`function PosEx(const SubStr: UnicodeString; const S: UnicodeString)
 : SizeInt`

Visibility: default

Description: `PosEx` returns the position of the first occurrence of the character `C` or the substring `SubStr` in the string `S`, starting the search at position `Offset` (default 1). If `C` or `SubStr` does not occur in `S` after the given `Offset`, zero is returned. The position `Offset` is also searched.

Errors: None.

See also: `NPos` ([1319](#)), `AnsiContainsText` ([1300](#)), `AnsiContainsStr` ([1299](#))

74.3.72 PosSet

Synopsis: Return the position in a string of any character out of a set of characters.

Declaration: `function PosSet(const c: TSysCharSet; const s: ansistring) : SizeInt`
`function PosSet(const c: string; const s: ansistring) : SizeInt`

Visibility: default

Description: `PosSet` returns the position in `s` of the first found character which is in the set `c`. If none of the characters in `c` is found in `s`, then 0 is returned.

Errors: None.

See also: `PosEx` ([1321](#)), `PosSetEx` ([1322](#)), `#rtl.system.pos` ([1515](#)), `RPosEx` ([1326](#))

74.3.73 PosSetEx

Synopsis: Return the position in a string of any character out of a set of characters, starting at a certain position.

Declaration: `function PosSetEx(const c: TSysCharSet; const s: ansistring;
count: Integer) : SizeInt
function PosSetEx(const c: string; const s: ansistring; count: Integer)
: SizeInt`

Visibility: default

Description: `PosSetEx` returns the position in `s` of the first found character which is in the set `c`, and starts searching at character position `Count`. If none of the characters in `c` is found in `s`, then 0 is returned.

Errors: None.

See also: `PosEx` ([1321](#)), `PosSet` ([1321](#)), `#rtl.system.pos` ([1515](#)), `RPosEx` ([1326](#))

74.3.74 RandomFrom

Synopsis: Choose a random string from an array of strings.

Declaration: `function RandomFrom(const AValues: Array of string) : string; Overload`

Visibility: default

Description: `RandomFrom` picks at random a valid index in the array `AValues` and returns the string at that position in the array.

Errors: None.

See also: `AnsiMatchStr` ([1302](#)), `AnsiMatchText` ([1302](#))

74.3.75 RemoveLeadingChars

Synopsis: Remove any leading characters in a set from a string.

Declaration: `procedure RemoveLeadingChars(var S: AnsiString; const CSet: TSysCharSet)
procedure RemoveLeadingChars(var S: UnicodeString;
const CSet: TSysCharSet)`

Visibility: default

Description: `Removeleadingchars` removes any starting characters from `S` that appear in the set `CSet`. It stops removing characters as soon as a character not in `CSet` is encountered. This is similar in behaviour to `TrimLeft` ([1784](#)) which used whitespace as the set.

Errors: None.

See also: `TrimLeft` ([1784](#)), `RemoveTrailingChars` ([1323](#)), `RemovePadChars` ([1322](#)), `TrimLeftSet` ([1331](#))

74.3.76 RemovePadChars

Synopsis: Remove any trailing or leading characters in a set from a string.

Declaration: `procedure RemovePadChars(var S: AnsiString; const CSet: TSysCharSet)`

Visibility: default

Description: `RemovePadChars` removes any leading trailing characters from `S` that appear in the set `CSet`, i.e. it starts with the last character and works its way to the start of the string, and it stops removing characters as soon as a character not in `CSet` is encountered. Then the same procedure is repeated starting from the beginning of the string. This is similar in behaviour to `Trim` (1783) which used whitespace as the set.

Errors: None.

See also: `Trim` (1783), `RemoveLeadingChars` (1322), `RemoveTrailingChars` (1323), `TrimSet` (1332), `TrimLeftSet` (1331), `TrimRightSet` (1332)

74.3.77 RemoveTrailingChars

Synopsis: Remove any trailing characters in a set from a string.

Declaration:

```
procedure RemoveTrailingChars(var S: AnsiString;
                             const CSet: TSysCharSet)
procedure RemoveTrailingChars(var S: UnicodeString;
                             const CSet: TSysCharSet)
```

Visibility: default

Description: `RemoveTrailingChars` removes any trailing characters from `S` that appear in the set `CSet`, i.e. it starts with the last character and works its way to the start of the string. It stops removing characters as soon as a character not in `CSet` is encountered. This is similar in behaviour to `TrimRight` (1785) which used whitespace as the set.

See also: `TrimRight` (1784), `RemoveLeadingChars` (1322), `TrimRightSet` (1332)

74.3.78 ReplaceStr

Synopsis: Replace strings case-sensitively.

Declaration:

```
function ReplaceStr(const AText: string; const AFromText: string;
                  const AToText: string) : string
```

Visibility: default

Description: `ReplaceStr` is a utility function that scans `AText` and replaces all occurrences of `AFromText` with `AToText` and returns the resulting string. It simply calls `StringReplace` (1759) with the appropriate options.

See also: `StringReplace` (1759), `ReplaceText` (1323)

74.3.79 ReplaceText

Synopsis: Replace strings case-insensitively.

Declaration:

```
function ReplaceText(const AText: string; const AFromText: string;
                   const AToText: string) : string
```

Visibility: default

Description: `ReplaceText` is a utility function that scans `AText` and replaces all occurrences of `AFromText` (case insensitive) with `AToText` and returns the resulting string. It simply calls `StringReplace` (1759) with the appropriate options.

See also: `StringReplace` (1759), `ReplaceText` (1323)

74.3.80 ResemblesText

Synopsis: Check whether 2 strings resemble each other.

Declaration: `function ResemblesText(const AText: string; const AOther: string)
: Boolean`

Visibility: default

Description: `ResemblesText` is an alias for `AnsiResemblesText` ([1303](#))

See also: `AnsiResemblesText` ([1303](#))

74.3.81 ReverseString

Synopsis: Reverse characters in a string.

Declaration: `function ReverseString(const AText: string) : string`

Visibility: default

Description: `ReverseString` returns a string, made up of the characters in string `AText`, in reverse order.

Errors: None.

See also: `RandomFrom` ([1322](#))

74.3.82 RightBStr

Synopsis: Copy a given number of characters (bytes), counting from the right of a string.

Declaration: `function RightBStr(const AText: AnsiString; const AByteCount: SizeInt)
: AnsiString`

Visibility: default

Description: `RightBStr` returns a string containing the rightmost `AByteCount` bytes from the string `AText`.
If `AByteCount` is larger than the length (in bytes) of `AText`, only as many bytes as available are returned.

Errors: None.

See also: `LeftBStr` ([1317](#)), `AnsiRightStr` ([1304](#)), `RightStr` ([1324](#)), `MidBStr` ([1318](#))

74.3.83 RightStr

Synopsis: Copy a given number of characters, counting from the right of a string.

Declaration: `function RightStr(const AText: AnsiString; const ACount: SizeInt)
: AnsiString
function RightStr(const AText: WideString; const ACount: SizeInt)
: WideString`

Visibility: default

Description: `RightStr` returns a string containing the rightmost `ACount` characters from the string `AText`.
If `ACount` is larger than the length (in characters) of `AText`, only as many characters as available are returned.

Errors: None.

See also: `LeftStr` ([1317](#)), `AnsiRightStr` ([1304](#)), `RightBStr` ([1324](#)), `MidStr` ([1318](#))

74.3.84 RomanToInt

Synopsis: Convert a string with a Roman number to it's decimal value.

Declaration: `function RomanToInt(const S: string;
 Strictness: TRomanConversionStrictness) : LongInt`

Visibility: default

Description: `RomanToInt` returns the decimal equivalent of the Roman numerals in the string `S`. Invalid characters are dropped from `S`. A negative numeral is supported as well. The level of error checking is determined by the `strictness` parameter, the values are described in the type `TRomanConversionStrictness` (1297).

Errors: On error, a `EConvertError` (1296) exception is raised.

See also: `TRomanConversionStrictness` (1297), `IntToRoman` (1316), `Hex2Dec` (1313), `Numb2Dec` (1320)

74.3.85 RomanToIntDef

Synopsis: Convert a roman numeral to an integer value.

Declaration: `function RomanToIntDef(const S: string; const ADefault: LongInt;
 Strictness: TRomanConversionStrictness) : LongInt`

Visibility: default

Description: `RomanToInt` converts the roman numeral in `S` to an integer and returns the integer value. The strictness of the conversion algorithm is determined by `Strictness`. If the conversion fails, `ADefault` is returned.

See also: `TRomanConversionStrictness` (1297), `TryRomanToInt` (1332), `RomanToInt` (1325), `IntToRoman` (1316)

74.3.86 RPos

Synopsis: Find last occurrence of substring or character in a string.

Declaration: `function RPos(c: AnsiChar; const S: AnsiString) : SizeInt; Overload
function RPos(c: UnicodeChar; const S: UnicodeString) : SizeInt
 ; Overload
function RPos(const Substr: AnsiString; const Source: AnsiString)
 : SizeInt; Overload
function RPos(const Substr: UnicodeString; const Source: UnicodeString)
 : SizeInt; Overload`

Visibility: default

Description: `RPos` looks in `S` for the character `C` or the string `SubStr`. It starts looking at the end of the string, and searches towards the beginning of the string. If a match is found, it returns the position of the match. If no match is found, 0 is returned.

See also: `RPosEx` (1326)

74.3.87 RPosEx

Synopsis: Find last occurrence substring or character in a string, starting at a certain position.

Declaration:

```

function RPosEx(C: AnsiChar; const S: AnsiString; offs: Cardinal)
    : SizeInt; Overload
function RPosEx(C: UnicodeChar; const S: UnicodeString; offs: Cardinal)
    : SizeInt; Overload
function RPosEx(const Substr: AnsiString; const Source: AnsiString;
    offs: Cardinal) : SizeInt; Overload
function RPosEx(const Substr: UnicodeString;
    const Source: UnicodeString; offs: Cardinal) : SizeInt
    ; Overload

```

Visibility: default

Description: RPos looks in S for the character C or the string SubStr. It starts looking at position Offs (counted from the start of the string), and searches towards the beginning of the string. If a match is found, it returns the position of the match. If no match is found, 0 is returned.

See also: RPos ([1325](#))

74.3.88 SearchBuf

Synopsis: Search a buffer for a certain string.

Declaration:

```

function SearchBuf(Buf: PAnsiChar; BufLen: SizeInt; SelStart: SizeInt;
    SelLength: SizeInt; SearchString: string;
    Options: TStringSearchOptions) : PAnsiChar
function SearchBuf(Buf: PAnsiChar; BufLen: SizeInt; SelStart: SizeInt;
    SelLength: SizeInt; SearchString: string) : PAnsiChar

```

Visibility: default

Description: SearchBuf searches the buffer Buf for the occurrence of SearchString. At most Buflen characters are searched, and the search is started at at SelStart when searching (up, the default), but starts at SelStart+SelLength when searching backward (the soDown option is given). If a match is found, a pointer to the position of the match is returned. SelStart is a zero-based offset from Buf and is the lower bound for search operations. The parameter Options ([1298](#)) specifies how the search is conducted. It is a set of the following options:

Table 74.5:

Option	Effect
soDown	Searches backward, starting at the end of the selection. Default is searching up.
soMatchCase	Observe case when searching. Default is to ignore case.
soWholeWord	Match only whole words. Default also returns parts of words.

The standard constant WordDelimiters ([1297](#)) is used to mark the boundaries of words.

The SelStart parameter is zero based.

Errors: Buflen must be the real length of the string, no checking on this is performed.

See also: FindPart ([1312](#)), ExtractWord ([1310](#)), ExtractWordPos ([1311](#)), ExtractSubStr ([1310](#)), IsWordPresent ([1317](#))

74.3.89 Soundex

Synopsis: Compute the soundex of a string.

Declaration: `function Soundex(const AText: string; ALength: TSoundexLength) : string`
`function Soundex(const AText: string) : string`

Visibility: default

Description: `Soundex` computes a soundex code for `AText`. The resulting code will at most have `ALength` characters. The soundex code is computed according to the US system of soundex computing, which may result in inaccurate results in other languages.

Note that `AText` may not contain null characters.

Errors: None.

See also: `SoundexCompare` (1327), `SoundexInt` (1327), `SoundexProc` (1328), `SoundexWord` (1328), `SoundexSimilar` (1328)

74.3.90 SoundexCompare

Synopsis: Compare soundex values of 2 strings.

Declaration: `function SoundexCompare(const AText: string; const AOther: string;`
`ALength: TSoundexLength) : Integer`
`function SoundexCompare(const AText: string; const AOther: string)`
`: Integer`

Visibility: default

Description: `SoundexCompare` computes the soundex codes of `AText` and `AOther` and feeds these to `CompareText`. It will return -1 if the soundex code of `AText` is less than the soundex code of `AOther`, 0 if they are equal, and 1 if the code of `AOther` is larger than the code of `AText`.

Errors: None.

See also: `Soundex` (1327), `SoundexInt` (1327), `SoundexProc` (1328), `SoundexWord` (1328), `SoundexSimilar` (1328)

74.3.91 SoundexInt

Synopsis: Soundex value as an integer.

Declaration: `function SoundexInt(const AText: string; ALength: TSoundexIntLength)`
`: Integer`
`function SoundexInt(const AText: string) : Integer`

Visibility: default

Description: `SoundexInt` computes the `Soundex` (1327) code (with length `ALength`, default 4) of `AText`, and converts the code to an integer value.

Errors: None.

See also: `Soundex` (1327), `SoundexCompare` (1327), `SoundexProc` (1328), `SoundexWord` (1328), `SoundexSimilar` (1328)

74.3.92 SoundexProc

Synopsis: Default AnsiResemblesText implementation.

Declaration: `function SoundexProc(const AText: string; const AOther: string)
: Boolean`

Visibility: default

Description: `SoundexProc` is the standard implementation for the `AnsiResemblesText` (1303) procedure: By default, `AnsiResemblesProc` is set to this function. It compares the soundex codes of `AOther` and `AText` and returns `True` if they are equal, or `False` if they are not.

Errors: None.

See also: `Soundex` (1327), `SoundexCompare` (1327), `SoundexInt` (1327), `SoundexWord` (1328), `SoundexSimilar` (1328)

74.3.93 SoundexSimilar

Synopsis: Check whether 2 strings have equal soundex values.

Declaration: `function SoundexSimilar(const AText: string; const AOther: string;
ALength: TSoundexLength) : Boolean
function SoundexSimilar(const AText: string; const AOther: string)
: Boolean`

Visibility: default

Description: `SoundexSimilar` returns `True` if the soundex codes (with length `ALength`) of `AText` and `AOther` are equal, and `False` if they are not.

Errors: None.

See also: `Soundex` (1327), `SoundexCompare` (1327), `SoundexInt` (1327), `SoundexProc` (1328), `SoundexWord` (1328), `Soundex` (1327)

74.3.94 SoundexWord

Synopsis: Calculate a word-sized soundex value.

Declaration: `function SoundexWord(const AText: string) : Word`

Visibility: default

Description: `SoundexInt` computes the `Soundex` (1327) code (with length 4) of `AText`, and converts the code to a word-sized value.

`AText` may not contain null characters.

Errors: None.

See also: `Soundex` (1327), `SoundexCompare` (1327), `SoundexInt` (1327), `SoundexProc` (1328), `SoundexSimilar` (1328)

74.3.95 SplitCommandLine

Synopsis: Split commandline in a series of separate arguments.

Declaration: `function SplitCommandLine(S: RawByteString) : TRawByteStringArray`
`function SplitCommandLine(S: UnicodeString) : TUnicodeStringArray`

Visibility: default

Description: `SplitCommandLine` parses `S` into words separated by whitespace, and returns a list of separate words (command-line arguments): When parsing it takes into account quoting with single (') or double quote (") characters: a word that contains whitespace can be enclosed in a quote character and will be considered a single word.

You can call this function with a `UnicodeString` or a `RawByteString`, the resulting array will have the same type of elements.

for example:

```
first "second argument" 'third argument'
```

Will result in an array of 3 elements

- first
- second argument
- third argument

See also: `TRawByteStringArray` ([1297](#)), `TUnicodeStringArray` ([1299](#))

74.3.96 SplitString

Synopsis: Split a string in words.

Declaration: `function SplitString(const S: string; const Delimiters: string)`
`: TRTLStringDynArray`

Visibility: default

Description: `SplitString` will split the string (`S`) using the characters in `Delimiters` as separator characters. The result contains all words separated by one of the characters in `Delimiters`.

This function is a simplified wrapper around `TStringHelper.Split` ([1296](#)).

See also: `TStringHelper.Split` ([1296](#))

74.3.97 StartsStr

Synopsis: Check whether one string starts with another.

Declaration: `function StartsStr(const ASubText: string; const AText: string)`
`: Boolean`

Visibility: default

Description: `StartsStr` checks whether `aText` starts with `aSubText` and returns `True` if it does. i.e. it returns true if the first characters of `aText` are `aSubText`. It follows that the length of `aText` must be at least the length of `aSubText`. The comparison is made case-sensitive. If you wish to compare case-insensitively, use `StartsText` ([1330](#)) instead.

See also: `AnsiStartsStr` ([1304](#)), `EndsStr` ([1309](#)), `StartsText` ([1330](#))

74.3.98 StartsWith

Synopsis: Check whether one text starts with another.

Declaration: `function StartsWith(const ASubText: string; const AText: string)
: Boolean`

Visibility: default

Description: `StartsWith` checks whether `aText` starts with `aSubText` and returns `True` if it does. i.e. it returns `true` if the first characters of `aText` are `aSubText`. It follows that the length of `aText` must be at least the length of `aSubText`. The comparison is made case-insensitive. If you wish to compare case-sensitively, use `StartsWithStr` (1329) instead.

See also: `AnsiStartsWith` (1304), `EndsWith` (1309), `StartsWithStr` (1329)

74.3.99 StringReplace

Synopsis: Optimized search-and-replace algorithm.

Declaration: `function StringReplace(const S: string; const OldPattern: string;
const NewPattern: string; Flags: TReplaceFlags;
out aCount: Integer;
Algorithm: TStringReplaceAlgorithm) : string
; Overload
function StringReplace(const S: string; const OldPattern: string;
const NewPattern: string; Flags: TReplaceFlags;
Algorithm: TStringReplaceAlgorithm) : string
; Overload`

Visibility: default

Description: `StringReplace` replaces one or all occurrences of `OldPattern` with `NewPattern` in the string `S`. The behaviour is controlled by `Flags` and the search mechanism may be optimized using `Algorithm`. A list of possible algorithms is specified in `TStringReplaceAlgorithm` (1298).

For `ansistrings`, this is an optimized version of the `SysUtils.StringReplace` (1759) algorithm. For `widestrings` or `unicodestrings`, the algorithm parameter is ignored and the default mechanism in `sysutils` is always used.

Errors: None.

See also: `SysUtils.StringReplace` (1759), `TStringReplaceAlgorithm` (1298)

74.3.100 StringsReplace

Synopsis: Replace occurrences of a set of strings to another set of strings.

Declaration: `function StringsReplace(const S: Ansistring;
OldPattern: Array of Ansistring;
NewPattern: Array of Ansistring;
Flags: TReplaceFlags) : string`

Visibility: default

Description: `StringsReplace` scans `S` for the occurrence of one of the strings in `OldPattern` and replaces it with the corresponding string in `NewPattern`. It takes into account `Flags`, which has the same meaning as in `StringReplace` (1759).

Corresponding strings are matched by location: the N-th string in `OldPattern` is replaced by the N-th string in `NewPattern`. Note that this means that the number of strings in both arrays must be the same.

Errors: If the number of strings in both arrays is different, then an exception is raised.

See also: `StringReplace` ([1759](#)), `TReplaceFlags` ([1637](#))

74.3.101 StuffString

Synopsis: Replace part of a string with another string.

Declaration: `function StuffString(const AText: string; AStart: Cardinal; ALength: Cardinal; const ASubText: string) : string`

Visibility: default

Description: `StuffString` returns a copy of `AText` with the segment starting at `AStart` with length `ALength`, replaced with the string `ASubText`. Basically it deletes the segment of `Atext` and inserts the new text in it's place.

Errors: No checking on the validity of the `AStart` and `ALength` parameters is done. Providing invalid values may result in access violation errors.

See also: `FindPart` ([1312](#)), `DelChars` ([1308](#)), `DelSpace` ([1308](#)), `ExtractSubStr` ([1310](#)), `DupeString` ([1309](#))

74.3.102 Tab2Space

Synopsis: Convert tab characters to a number of spaces.

Declaration: `function Tab2Space(const S: string; Numb: Byte) : string`

Visibility: default

Description: `Tab2Space` returns a copy of `S` with all tab characters (ASCII character 9) converted to `Numb` spaces.

Errors: None.

See also: `StuffString` ([1331](#)), `FindPart` ([1312](#)), `ExtractWord` ([1310](#)), `DelChars` ([1308](#)), `DelSpace` ([1308](#)), `DelSpace1` ([1308](#)), `DupeString` ([1309](#))

74.3.103 TrimLeftSet

Synopsis: Remove any leading characters in a set from a string and returns the result.

Declaration: `function TrimLeftSet(const S: string; const CSet: TSysCharSet) : string`

Visibility: default

Description: `TrimLeftSet` performs the same action as `RemoveLeadingChars` ([1322](#)), but returns the resulting string.

Errors: None.

See also: `TrimLeft` ([1784](#)), `RemoveLeadingChars` ([1322](#)), `RemoveTrailingChars` ([1323](#)), `RemovePadChars` ([1322](#)), `TrimSet` ([1332](#)), `TrimRightSet` ([1332](#))

74.3.104 TrimRightSet

Synopsis: Remove any trailing characters in a set from a string and returns the result.

Declaration: `function TrimRightSet(const S: string; const CSet: TSysCharSet) : string`

Visibility: default

Description: `TrimLeftSet` performs the same action as `RemoveTrailingChars` (1323), but returns the resulting string.

Errors: None.

See also: `TrimRight` (1785), `RemoveLeadingChars` (1322), `RemoveTrailingChars` (1323), `RemovePadChars` (1322), `TrimSet` (1332), `TrimLeftSet` (1331)

74.3.105 TrimSet

Synopsis: Remove any leading or trailing characters in a set from a string and returns the result.

Declaration: `function TrimSet(const S: string; const CSet: TSysCharSet) : string`

Visibility: default

Description: `TrimSet` performs the same action as `RemovePadChars` (1322), but returns the resulting string.

Errors: None.

See also: `Trim` (1783), `RemoveLeadingChars` (1322), `RemoveTrailingChars` (1323), `RemovePadChars` (1322), `TrimRightSet` (1332), `TrimLeftSet` (1331)

74.3.106 TryRomanToInt

Synopsis: Try to convert a roman numeral to an integer value.

Declaration: `function TryRomanToInt(S: string; out N: LongInt;
 Strictness: TRomanConversionStrictness) : Boolean`

Visibility: default

Description: `TryRomanToInt` will try to convert the roman numeral in `S` to an integer and returns the integer value in `N`. The strictness of the conversion algorithm is determined by `Strictness`. If the conversion succeeds, then `True` is returned, or else `False`.

See also: `TRomanConversionStrictness` (1297), `RomanToIntDef` (1325), `RomanToInt` (1325), `IntToRoman` (1316)

74.3.107 WordCount

Synopsis: Count the number of words in a string.

Declaration: `function WordCount(const S: string; const WordDelims: TSysCharSet)
 : SizeInt`

Visibility: default

Description: `WordCount` returns the number of words in the string `S`. A word is a non-empty string of characters bounded by one of the characters in `WordDelims`.

The predefined `StdWordDelims` (1297) constant can be used for the `WordDelims` argument.

Errors: None.

See also: [WordPosition \(1333\)](#), [StdWordDelims \(1297\)](#), [ExtractWord \(1310\)](#), [ExtractWordPos \(1311\)](#)

74.3.108 WordPosition

Synopsis: Search position of Nth word in a string.

Declaration:

```
function WordPosition(const N: Integer; const S: string;
                      const WordDelims: TSysCharSet) : SizeInt
```

Visibility: default

Description: `WordPosition` returns the position (in characters) of the N-th word in the string S. A word is a non-empty string of characters bounded by one of the characters in `WordDelims`. If N is out of range, zero is returned.

The predefined `StdWordDelims` ([1297](#)) constant can be used for the `WordDelims` argument.

Errors: None

See also: [WordCount \(1332\)](#), [StdWordDelims \(1297\)](#), [ExtractWord \(1310\)](#), [ExtractWordPos \(1311\)](#)

74.3.109 XorDecode

Synopsis: Decode a string encoded with `XorEncode` ([1333](#)).

Declaration:

```
function XorDecode(const Key: Ansistring; const Source: Ansistring)
                : Ansistring
```

Visibility: default

Description: `XorDecode` decodes `Source` and returns the original string that was encrypted using `XorEncode` ([1333](#)) with key `Key`. If a different key is used than the key used to encode the string, the result will be unreadable.

Errors: If the string `Source` is not a valid `XorEncode` result (e.g. contains non-numerical characters), then a `EConversionError` exception will be raised.

See also: [XorEncode \(1333\)](#), [XorString \(1334\)](#)

74.3.110 XorEncode

Synopsis: Encode a string by XOR-ing its characters using characters of a given key, representing the result as hex values.

Declaration:

```
function XorEncode(const Key: Ansistring; const Source: Ansistring)
                : Ansistring
```

Visibility: default

Description: `XorEncode` encodes the string `Source` by XOR-ing each character in `Source` with the corresponding character in `Key` (repeating `Key` as often as necessary) and representing the resulting ASCII code as a hexadecimal number (of length 2). The result is therefore twice as long as the original string, and every 2 bytes represent an ASCII code.

Feeding the resulting string with the same key `Key` to the `XorDecode` ([1333](#)) function will result in the original `Source` string.

This function can be used e.g. to trivially encode a password in a configuration file.

Errors: None.

See also: [XorDecode \(1333\)](#), [XorString \(1334\)](#), [Hex2Dec \(1313\)](#)

74.3.111 XorString

Synopsis: Encode a string by XOR-ing its characters using characters of a given key.

Declaration: `function XorString(const Key: ShortString; const Src: ShortString)
: ShortString`

Visibility: default

Description: `XorString` encodes the string `Src` by XOR-ing each character in `Source` with the corresponding character in `Key`, repeating `Key` as often as necessary. The resulting string may contain unreadable characters and may even contain null characters. For this reason it may be a better idea to use the [XorEncode \(1333\)](#) function instead, which will represent each resulting ASCII code as a hexadecimal number (of length 2).

Feeding the result again to `XorString` with the same `Key`, will result in the original string `Src`.

Errors: None.

See also: [XorEncode \(1333\)](#), [XorDecode \(1333\)](#)

Chapter 75

Reference for unit 'System'

75.1 Overview

The system unit contains the standard supported functions of Free Pascal. It is the same for all platforms. Basically it is the same as the system unit provided with Borland or Turbo Pascal.

Functions are listed in alphabetical order. Arguments of functions or procedures that are optional are put between square brackets.

The predefined constants and variables are listed in the first section. The second section contains an overview of all functions, grouped by functionality, and the last section contains the supported functions and procedures.

Remark The RTL documentation is generated from the actual sources. This means that for some CPU-dependent or platform-dependent types, the shown definition depends on the platform and CPU for which the documentation is generated. (which is normally the Linux platform and i386 CPU)

75.2 A list of managed types.

Some of the basic pascal types are managed. This means that their lifetime is treated specially by the compiler: Variables of a managed type:

- Are implicitly initialized at the beginning of their scope. Usually this means the memory area reserved for them is zeroed out.
- Use some form of reference counting when they are assigned to one another during their lifetime.
- Are implicitly finalized at the end of their scope. Usually this means decreasing a reference count and freeing the dynamic memory occupied by the variable if the reference count returns zero.

The following types are managed:

AnsiString The one-byte character string uses a reference count.

UnicodeString The two-byte character string uses a reference count

WideString Is equal to Unicodestring on non-windows platforms, is managed specially on windows.

Dynamic arrays Are reference counted.

Interfaces COM interfaces are reference counted through the mechanisms in `IInterface`. CORBA interfaces are not managed.

Additionally, structured types such as arrays, records, objects and classes containing one of the above types automatically become managed too. Note that for classes this does not mean that the class instance itself becomes managed, but when creating/destroying the class through the constructor and destructor, the fields in a class that are of managed type will be initialized and finalized.

For more information about the exact workings for each type, please consult the language reference guide.

75.3 Unicode and codepage support.

The system unit works with Short-, Ansi-, and UnicodeString routines for all string related operations.

Ansistrings are code-page aware, which means that code page information is associated with them. For most routines, the support for converting these code pages is natural. For some routines, care must be taken when converting from codepage-aware strings to widestring.

The codepage conversion support is influenced by the following variables:

Table 75.1:

Name	Description
DefaultSystemCodePage (1415)	Actual code page to use when CP_ACP (1343) is encountered
DefaultUnicodeCodePage (1415)	Code page for new Unicode strings
DefaultFileSystemCodePage (1414)	Codepage to use when sending strings to single-byte OS file system routines.
DefaultRTLFileSystemCodePage (1414)	Codepage to use when receiving strings from single-byte OS file system routines.

The windows code page identifiers are used. There are 3 special codepage identifiers:

Table 75.2:

Name	Description
CP_ACP (1343)	Currently set default system codepage
CP_OEMCP (1343)	OEM (console) code page (only on windows)
CP_NONE (1343)	Indicates absence of code page information for a string
DefaultRTLFileSystemCodePage (1414)	

The following routines may perform code page conversions:

Table 75.3:

Name	Description
LowerCase (1506)	Return lowercase version of a string.
UpCase (1567)	Convert a string to all uppercase.
GetDir (1473)	Return the current directory.
MkDir (1507)	Create a new directory.
ChDir (1436)	Change current working directory.
RmDir (1526)	Remove directory when empty.
Assign (1423)	Assign a name to a file.
Erase (1458)	Delete a file from disk.
Rename (1524)	Rename file on disk.
Read (1519)	Read from a text file into variable.
ReadLn (1521)	Read from a text file into variable and goto next line.
Write (1574)	Write variable to a text file or standard output.
WriteLn (1575)	Write variable to a text file or standard output and append newline.
ReadStr (1521)	Read variables from a string.
WriteStr (1576)	Write variables to a string.
Insert (1493)	Insert one string or dynamic array in another.
Copy (1445)	Copy part of a string or dynamic array.
Delete (1449)	Delete elements (characters) from a string or dynamic array.
SetString (1541)	Set length of a string and copy buffer.

All these routines exist also in Unicode versions.

Note that for conversion of codepages and Unicode strings, a Unicode manager must be present. On windows, the system is used for this. On Unix, one of the fpwidingstring or cwstring units must be used.

75.4 Miscellaneous functions.

Functions that do not belong in one of the other categories.

Table 75.4:

Name	Description
Assert (1423)	Conditionally abort program with error
Break (1432)	Abort current loop
Continue (1444)	Next cycle in current loop
Exclude (1459)	Exclude an element from a set
Exit (1460)	Exit current function or procedure
Include (1485)	Include an element into a set
LongJump (1505)	Jump to execution point
Ord (1512)	Return ordinal value of enumerated type
Pred (1517)	Return previous value of ordinal type
SetJump (1537)	Mark execution point for jump
SizeOf (1546)	Return size of variable or type
Succ (1554)	Return next value of ordinal type

75.5 Operating System functions.

Functions that are connected to the operating system.

Table 75.5:

Name	Description
Chdir (1436)	Change working directory
Getdir (1473)	Return current working directory
Halt (1481)	Halt program execution
Paramcount (1513)	Number of parameters with which program was called
Paramstr (1514)	Retrieve parameters with which program was called
Mkdir (1507)	Make a directory
Rmdir (1526)	Remove a directory
Runerror (1532)	Abort program execution with error condition

75.6 String handling.

All things connected to string handling.

Table 75.6:

Name	Description
BinStr (1430)	Construct binary representation of integer
Chr (1437)	Convert ASCII code to character
Concat (1443)	Concatenate two strings
Copy (1445)	Copy part of a string
Delete (1449)	Delete part of a string
HexStr (1481)	Construct hexadecimal representation of integer
Insert (1493)	Insert one string in another
Length (1500)	Return length of string
Lowercase (1506)	Convert string to all-lowercase
OctStr (1511)	Construct octal representation of integer
Pos (1515)	Calculate position of one string in another
SetLength (1538)	Set length of a string
SetString (1541)	Set contents and length of a string or dynamic array
Str (1549)	Convert number to string representation
StringOfChar (1552)	Create string consisting of a number of characters
Uppcase (1567)	Convert string to all-uppercase
Val (1570)	Convert string to number

75.7 Mathematical routines.

Functions connected to calculating and converting numbers.

Table 75.7:

Name	Description
Abs (1418)	Calculate absolute value
Arctan (1422)	Calculate inverse tangent
Cos (1446)	Calculate cosine of angle
Dec (1447)	Decrease value of variable
Exp (1462)	Exponentiate
Frac (1471)	Return fractional part of floating point value
Hi (1482)	Return high byte/word of value
Inc (1484)	Increase value of variable
Int (1493)	Calculate integer part of floating point value
Ln (1502)	Calculate logarithm
Lo (1502)	Return low byte/word of value
Odd (1511)	Is a value odd or even ?
Pi (1514)	Return the value of pi
Random (1518)	Generate random number
Randomize (1519)	Initialize random number generator
Round (1529)	Round floating point value to nearest integer number
Sin (1545)	Calculate sine of angle
Sqr (1548)	Calculate the square of a value
Sqrt (1548)	Calculate the square root of a value
Swap (1555)	Swap high and low bytes/words of a variable
Trunc (1561)	Truncate a floating point value

75.8 Memory management functions.

Functions concerning memory issues.

Table 75.8:

Name	Description
Addr (1420)	Return address of variable
Assigned (1428)	Check if a pointer is valid
CompareByte (1438)	Compare 2 memory buffers byte per byte
CompareChar (1439)	Compare 2 memory buffers byte per byte
CompareDWord (1441)	Compare 2 memory buffers byte per byte
CompareWord (1442)	Compare 2 memory buffers byte per byte
CSeg (1447)	Return code segment
Dispose (1450)	Free dynamically allocated memory
DSeg (1452)	Return data segment
FillByte (1465)	Fill memory region with 8-bit pattern
FillChar (1465)	Fill memory region with certain character
FillDWord (1466)	Fill memory region with 32-bit pattern
FillQWord (??)	Fill memory region with 64-bit pattern
FillWord (1466)	Fill memory region with 16-bit pattern
Freemem (1471)	Release allocated memory
Getmem (1474)	Allocate new memory
GetMemoryManager (1475)	Return current memory manager
High (1483)	Return highest index of open array or enumerated
IndexByte (1485)	Find byte-sized value in a memory range
IndexChar (1486)	Find char-sized value in a memory range
IndexDWord (1487)	Find DWord-sized (32-bit) value in a memory range
IndexQWord (1488)	Find QWord-sized value in a memory range
IndexWord (1488)	Find word-sized value in a memory range
IsMemoryManagerSet (1499)	Is the memory manager set
MemSize (1507)	Get size of allocation
Low (1506)	Return lowest index of open array or enumerated
Move (1508)	Move data from one location in memory to another
MoveChar0 (1508)	Move data till first zero character
New (1509)	Dynamically allocate memory for variable
Ofs (1512)	Return offset of variable
Ptr (1517)	Combine segment and offset to pointer
ReAllocMem (1523)	Resize a memory block on the heap
Seg (1536)	Return segment
SetMemoryManager (1539)	Set a memory manager
Sptr (1547)	Return current stack pointer
SSeg (1549)	Return stack segment register value

75.9 File handling functions.

Functions concerning input and output from and to file.

Table 75.9:

Name	Description
Append (1421)	Open a file in append mode
Assign (1423)	Assign a name to a file
Blockread (1431)	Read data from a file into memory
Blockwrite (1432)	Write data from memory to a file
Close (1437)	Close a file
Eof (1456)	Check for end of file
Eoln (1457)	Check for end of line
Erase (1458)	Delete file from disk
Filepos (1463)	Position in file
Filesize (1464)	Size of file
Flush (1469)	Write file buffers to disk
IOresult (1497)	Return result of last file IO operation
Read (1519)	Read from file into variable
Readln (1521)	Read from file into variable and goto next line
Rename (1524)	Rename file on disk
Reset (1524)	Open file for reading
Rewrite (1525)	Open file for writing
Seek (1534)	Set file position
SeekEof (1535)	Set file position to end of file
SeekEoln (1536)	Set file position to end of line
SetTextBuf (1542)	Set size of file buffer
Truncate (1562)	Truncate the file at position
Write (1574)	Write variable to file
WriteLn (1575)	Write variable to file and append newline

75.10 Run-Time Error behaviour.

The system unit handles errors by default by generating a run-time error, and halting the program with an exit code equal to the run-time error number.

This behaviour changes when the SysUtils (1609) unit is used. In that case, all run-time errors are converted to exceptions: most run-time errors have their own exception class.

If these exceptions are caught, the program code decides what to do with it. If the exception is not caught, the program will exit through the default exception handler.

When the system unit documentation refers to run-time errors, the above should be kept in mind.

75.11 Constants, types and variables

75.11.1 Constants

```
AbstractErrorProc : TAbstractErrorProc = Nil
```

If set, the `AbstractErrorProc` constant is used when an abstract error occurs. If it is not set, then the standard error handling is done: A stack dump is performed, and the program exits with error code 211.

The `SysUtils` unit sets this procedure and raises an exception in its handler.

```
AllFilesMask = '*'
```

AllFilesMask is the wildcard that can be used to return all files in a directory. Do not assume that this is '*' or '*.*' based on the platform only. The actual value on DOS/Windows based systems can be influenced by e.g. LFNSupport ([1358](#)).

```
AllowDirectorySeparators : Set of Char = ['\','/']
```

AllowDirectorySeparators is the set of characters which are considered directory separators by the RTL units. By default, this is set to the most common directory separators: forward slash and backslash, so routines will work in a cross-platform manner, no matter which character was used:

```
AllowDirectorySeparators : set of char = ['\','/'];
```

If a more strict behaviour is desired, then AllowDirectorySeparators can be set to the only character allowed on the current operating system, and all RTL routines that handle filenames (splitting filenames, extracting parts of the filename and so on) will use that character only.

```
AllowDriveSeparators : Set of Char = []
```

AllowDriveSeparators are the characters which are considered to separate the drive part from the directory part in a filename. This will be an empty set on systems that do not support drive letters. Other systems (Dos, Windows and OS/2) will have the colon (:) character as the only member of this set.

```
AssertErrorProc : TAssertErrorProc = @ SysAssert
```

If set, the AssertErrorProc constant is used when an assert error occurs. If it is not set, then the standard assert error handling is performed:

- The assertion error message is printed.
- The location of the assertion is printed.
- a stack dump is printed.
- The program exits with error code 227.

The SysUtils unit sets this procedure and raises an exception in its handler.

```
BacktraceStrFunc : TBacktraceStrFunc = @ SysBacktraceStr
```

This handler is called to get a standard format for the backtrace routine.

```
CatchAllExceptions = PtrInt(- 1)
```

Mask indicating all exception kinds.

```
cExceptionFrame = 1
```

cExceptionFrame indicates an except frame.

```
cFinalizeFrame = 2
```

`cFinalizeFrame` indicates a finally frame.

`CP_ACP` = 0

`CP_ACP` is the default Windows codepage identifier.

`CP_ASCII` = 20127

`CP_ASCII` is the Windows ASCII encoding codepage identifier.

`CP_NONE` = \$FFFF

`CP_NONE` is used when no code page information is available.

`CP_OEMCP` = 1

`CP_ACP` is the default Windows OEM (MS-DOS) codepage identifier.

`CP_UTF16` = 1200

`CP_UTF16` is the default Windows Unicode codepage identifier (little endian).

`CP_UTF16BE` = 1201

`CP_UTF16BE` is the Windows Unicode codepage identifier (big endian).

`CP_UTF7` = 65000

`CP_UTF7` is the Windows Unicode 7-Bit encoding codepage identifier.

`CP_UTF8` = 65001

`CP_UTF8` is the Windows Unicode 8-Bit encoding codepage identifier.

`CtrlZMarksEOF` : Boolean = False

`CtrlZMarksEOF` indicates whether on this system, an CTRL-Z character (ordinal 26) in a file marks the end of the file. This is False on most systems apart from DOS and Windows.

To get DOS/Windows-compatible behaviour, this constant can be set to True

`Default8087CW` : Word = \$1332

`DefaultMXCSR` : DWord = \$1900

`DefaultStackSize` = 4 * 1024 * 1024

Default size for a new thread's stack (4MiB by default).

`DefaultTextLineBreakStyle` : TTextLineBreakStyle = tlbsLF

`DefaultTextLineBreakStyle` contains the default OS setting for the `TTextLineBreakStyle` (1399) type. It is initialized by the system unit, and is used to determine the default line ending when writing to text files.

This constant is part of a set of constants that describe the OS characteristics. These constants should be used instead of hardcoding OS characteristics.

```
DirectorySeparator = '/'
```

`DirectorySeparator` is the character used by the current operating system to separate directory parts in a pathname. This constant is system dependent, and should not be set.

This constant is part of a set of constants that describe the OS characteristics. These constants should be used instead of hardcoding OS characteristics.

```
DriveSeparator = ''
```

On systems that support drive letters, the `DriveSeparator` constant denotes the character that separates the drive indicator from the directory part in a filename path.

This constant is part of a set of constants that describe the OS characteristics. These constants should be used instead of hardcoding OS characteristics.

```
ErrorAddr : CodePointer = Nil
```

Address where the last error occurred.

```
ErrorCode : Word = 0
```

Last error code.

```
ErrorProc : TErrorProc = Nil
```

If set, the `ErrorProc` constant is used when a run-time error occurs. If it is not set, then the standard error handling is done: a stack dump is performed, and the program exits with the indicated error code.

The `SysUtils` unit sets this procedure and raises an exception in its handler.

```
ExceptClsProc : Pointer = Nil
```

`ExceptClsProc` is used in SEH (Structured Exception Support) to convert OS exception information to FPC exception classes when filtering exceptions. It is set e.g. by the `sysutils` unit. If it is not set, the exception is not handled.

```
ExceptObjProc : Pointer = Nil
```

`ExceptObjProc` is used in SEH (Structured Exception Support) to convert OS exception information to FPC exceptions. It is set e.g. by the `sysutils` unit. If it is not set, a run-time error results when OS exceptions are intercepted.

```
ExceptProc : TExceptProc = Nil
```

This constant points to the current exception handling procedure. This routine is called when an unhandled exception occurs, i.e. an exception that is not stopped by a except block.

If the handler is not set, the RTL will emit a run-time error 217 when an unhandler exception occurs.

It is set by the sysutils ([1609](#)) unit.

```
ExitProc : CodePointer = Nil
```

Exit procedure pointer.

```
ExtensionSeparator = '.'
```

`ExtensionSeparator` is the character which separates the filename from the file extension. On all current platforms, this is the . (dot) character. All RTL filename handling routines use this constant.

```
E_NOINTERFACE = hresult($80004002)
```

Interface call result: Error: not an interface.

```
E_NOTIMPL = hresult($80004001)
```

Interface call result: Interface not implemented.

```
E_UNEXPECTED = hresult($8000FFFF)
```

Interface call result: Unexpected error.

```
FileMode : Byte = 2
```

`FileMode` determines how untyped files are opened for reading with `Reset` ([1524](#)). It can have the following values:

- 0** open file readonly
- 1** open file write only
- 2** open file read/write

```
FileNameCasePreserving : Boolean = True
```

`FileNameCasePreserving` is `True` if case of letters in file and directory entries is preserved and may be later retrieved exactly as supplied when creating or renaming these entries. Note that this may depend on the file system: Unix operating systems that access a DOS or Windows partition will have this constant set to true, but when writing to the DOS partition, all letters may be automatically converted to uppercase.

This constant is part of a set of constants that describe the OS characteristics. These constants should be used instead of hardcoding OS characteristics.

```
FileNameCaseSensitive : Boolean = True
```

`FileNameCaseSensitive` is `True` if case is important when using filenames on the current OS. In this case, the OS will treat files with different cased names as different files. Note that this may depend on the file system: Unix operating systems that access a DOS or Windows partition will have this constant set to true, but when writing to the DOS partition, the casing is ignored.

This constant is part of a set of constants that describe the OS characteristics. These constants should be used instead of hardcoding OS characteristics.

`filerecnamelength` = 255

`filerecnamelength` is the maximum filename size for untyped files.

`float_flag_denormal` = `exDenormalized`

IEC/IEEE floating-point exception flag: ?

`float_flag_divbyzero` = `exZeroDivide`

IEC/IEEE floating-point exception flag: Division by zero error.

`float_flag_inexact` = `exPrecision`

IEC/IEEE floating-point exception flag: ?

`float_flag_invalid` = `exInvalidOp`

IEC/IEEE floating-point exception flag: Invalid operation error.

`float_flag_overflow` = `exOverflow`

IEC/IEEE floating-point exception flag: Overflow error.

`float_flag_underflow` = `exUnderflow`

IEC/IEEE floating-point exception flag: Underflow error.

`float_round_down` = `rmDown`

Round down.

`float_round_nearest_even` = `rmNearest`

Round to nearest even number.

`float_round_to_zero` = `rmTruncate`

Round in the direction of zero (down for positive, up for negative).

`float_round_up` = `rmUp`

Round up.

`fmAppend = $D7B4`

File mode: File is open for writing, appending to the end.

`fmClosed = $D7B0`

File mode: File is closed.

`fmInOut = $D7B3`

File mode: File is open for reading and writing.

`fmInput = $D7B1`

File mode: File is open for reading.

`fmOutput = $D7B2`

File mode: File is open for writing.

`FPC_EXCEPTION = 1`

`FPC_EXCEPTION` is a constant describing a type of exception.

`fpc_in_abs_long = 64`

Internal ABS function.

`fpc_in_abs_real = 127`

FPC compiler internal procedure index: `abs` (real).

`fpc_in_addr_x = 42`

FPC compiler internal procedure index: `addr`.

`fpc_in_aligned_x = 80`

FPC compiler internal procedure index: `aligned`.

`fpc_in_and_assign_x_y = 86`

`fpc_in_arctan_real = 130`

FPC compiler internal procedure index: `arctan` (real).

`fpc_in_assert_x_y = 41`

FPC compiler internal procedure index: `assert`.

`fpc_in_assigned_x` = 19

FPC compiler internal procedure index: assigned.

`fpc_in_bitsizeof_x` = 61

FPC compiler internal procedure index: bitsizeof.

`fpc_in_box_x` = 77

FPC compiler internal procedure index: box.

`fpc_in_break` = 39

FPC compiler internal procedure index: break.

`fpc_in_bsf_x` = 74

FPC compiler internal procedure index: bsf_x.

`fpc_in_bsr_x` = 75

FPC compiler internal procedure index: bsr_x.

`fpc_in_chr_byte` = 7

FPC compiler internal procedure index: chr.

`fpc_in_concat_x` = 18

FPC compiler internal procedure index: concat.

`fpc_in_const_abs` = 101

FPC compiler internal procedure index: abs.

`fpc_in_const_odd` = 102

FPC compiler internal procedure index: sqr.

`fpc_in_const_ptr` = 103

FPC compiler internal procedure index: sqr.

`fpc_in_const_sqr` = 100

FPC compiler internal procedure index: sqr.

`fpc_in_const_swap_long` = 105

FPC compiler internal procedure index: swap (long).

`fpc_in_const_swap_qword = 108`

FPC compiler internal procedure index: swap (qword).

`fpc_in_const_swap_word = 104`

FPC compiler internal procedure index: swap (word).

`fpc_in_continue = 40`

FPC compiler internal procedure index: continue.

`fpc_in_copy_x = 49`

FPC compiler internal procedure index: copy.

`fpc_in_cos_real = 125`

FPC compiler internal procedure index: cos (real).

`fpc_in_cpu_first = 10000`

`fpc_in_cycle = 52`

FPC compiler internal procedure index: cycle.

`fpc_in_dec_x = 36`

FPC compiler internal procedure index: dec.

`fpc_in_default_x = 76`

FPC compiler internal procedure index: default.

`fpc_in_delete_x_y_z = 83`

`fpc_in_dispose_x = 47`

FPC compiler internal procedure index: dispose.

`fpc_in_exclude_x_y = 38`

FPC compiler internal procedure index: exclude.

`fpc_in_exit = 48`

FPC compiler internal procedure index: exit.

`fpc_in_exp_real = 124`

FPC internal compiler routine: `in_exp_real`.

`fpc_in_faraddr_x` = 97

`fpc_in_fillchar_x` = 55

FPC internal compiler routine: `in_fillchar_x`.

`fpc_in_finalize_x` = 45

FPC compiler internal procedure index: `finalize`.

`fpc_in_fma_double` = 134

FPC compiler internal procedure index: `fma (double)`.

`fpc_in_fma_extended` = 135

FPC compiler internal procedure index: `fma (extended)`.

`fpc_in_fma_float128` = 136

FPC compiler internal procedure index: `fma (float 128)`.

`fpc_in_fma_single` = 133

FPC compiler internal procedure index: `fma (single)`.

`fpc_in_frac_real` = 122

FPC internal compiler routine: `in_frac_real`.

`fpc_in_get_caller_addr` = 57

FPC internal compiler routine: `in_get_caller_addr`.

`fpc_in_get_caller_frame` = 58

FPC internal compiler routine: `in_get_caller_frame`.

`fpc_in_get_frame` = 56

FPC internal compiler routine: `in_get_frame`.

`fpc_in_high_x` = 28

FPC compiler internal procedure index: `high`.

`fpc_in_hi_long` = 4

FPC compiler internal procedure index: hi (long).

`fpc_in_hi_qword = 107`

FPC compiler internal procedure index: hi (qword).

`fpc_in_hi_word = 2`

FPC compiler internal procedure index: hi (word).

`fpc_in_include_x_y = 37`

FPC compiler internal procedure index: include.

`fpc_in_inc_x = 35`

FPC compiler internal procedure index: inc.

`fpc_in_initialize_x = 50`

FPC compiler internal procedure index: initialize.

`fpc_in_insert_x_y_z = 82`

`fpc_in_int_real = 123`

FPC internal compiler routine: in_int_real.

`fpc_in_leave = 51`

FPC compiler internal procedure index: leave.

`fpc_in_length_string = 6`

FPC compiler internal procedure index: length.

`fpc_in_ln_real = 131`

FPC compiler internal procedure index: ln (real).

`fpc_in_low_x = 27`

FPC compiler internal procedure index: low.

`fpc_in_lo_long = 3`

FPC compiler internal procedure index: lo (long).

`fpc_in_lo_qword = 106`

FPC compiler internal procedure index: lo (qword).

`fpc_in_lo_word = 1`

FPC compiler internal procedure index: lo (word).

`fpc_in_mmx_pcmpeqb = 200`

FPC compiler internal procedure index: MMX.

`fpc_in_mmx_pcmpeqd = 202`

FPC compiler internal procedure index: MMX.

`fpc_in_mmx_pcmpeqw = 201`

FPC compiler internal procedure index: MMX.

`fpc_in_mmx_pcmpgtb = 203`

FPC compiler internal procedure index: MMX.

`fpc_in_mmx_pcmpgtd = 205`

FPC compiler internal procedure index: MMX.

`fpc_in_mmx_pcmpgtw = 204`

FPC compiler internal procedure index: MMX.

`fpc_in_move_x = 54`

FPC internal compiler routine: in_move_x.

`fpc_in_neg_assign_x = 94`

`fpc_in_new_x = 46`

FPC compiler internal procedure index: new.

`fpc_in_not_assign_x = 95`

`fpc_in_ofs_x = 21`

FPC compiler internal procedure index: ofs.

`fpc_in_ord_x = 5`

FPC compiler internal procedure index: ord.

`fpc_in_or_assign_x_y` = 87

`fpc_in_pack_x_y_z` = 59

FPC compiler internal procedure index: `pack`.

`fpc_in_pi_real` = 126

FPC internal compiler routine: `in_pi_real`.

`fpc_in_popcnt_x` = 79

FPC compiler internal procedure index: `popcnt`.

`fpc_in_pred_x` = 30

FPC compiler internal procedure index: `pred`.

`fpc_in_prefetch_var` = 109

FPC compiler internal procedure index: `prefetch`.

`fpc_in_readln_x` = 17

FPC compiler internal procedure index: `readln`.

`fpc_in_readstr_x` = 63

Internal read string procedure.

`fpc_in_read_x` = 16

FPC compiler internal procedure index: `read`.

`fpc_in_reset_typedfile` = 32

FPC compiler internal procedure index: `reset`.

`fpc_in_reset_typedfile_name` = 84

`fpc_in_reset_x` = 25

FPC compiler internal procedure index: `reset`.

`fpc_in_rewrite_typedfile` = 33

FPC compiler internal procedure index: `rewrite`.

`fpc_in_rewrite_typedfile_name` = 85

fpc_in_rewrite_x = 26

FPC compiler internal procedure index: rewrite.

fpc_in_rol_assign_x_y = 92

fpc_in_rol_x = 67

fpc_in_rol_x_x = 68

fpc_in_ror_assign_x_y = 93

fpc_in_ror_x = 65

fpc_in_ror_x_x = 66

fpc_in_round_real = 121

FPC internal compiler routine: in_round_real.

fpc_in_sar_assign_x_y = 89

fpc_in_sar_x = 73

FPC compiler internal procedure index: sar_x.

fpc_in_sar_x_y = 72

FPC compiler internal procedure index: sar_x_y.

fpc_in_seg_x = 29

FPC compiler internal procedure index: seg.

fpc_in_setlength_x = 44

FPC compiler internal procedure index: setlength.

fpc_in_setstring_x_y_z = 81

FPC compiler internal procedure index: setstring.

fpc_in_settextbuf_file_x = 34

FPC compiler internal procedure index: settextbuf.

`fpc_in_shl_assign_x_y` = 90

`fpc_in_shr_assign_x_y` = 91

`fpc_in_sin_real` = 132

FPC compiler internal procedure index: `sin` (real).

`fpc_in_sizeof_x` = 22

FPC compiler internal procedure index: `sizeof`.

`fpc_in_slice` = 53

FPC internal compiler routine: `in_slice`.

`fpc_in_sqrt_real` = 129

FPC compiler internal procedure index: `sqrt` (real).

`fpc_in_sqr_real` = 128

FPC compiler internal procedure index: `sqr` (real).

`fpc_in_str_x_string` = 20

FPC compiler internal procedure index: `str`.

`fpc_in_succ_x` = 31

FPC compiler internal procedure index: `succ`.

`fpc_in_trunc_real` = 120

FPC internal compiler routine: `in_trunc_real`.

`fpc_in_typeinfo_x` = 43

FPC compiler internal procedure index: `typeinfo`.

`fpc_in_typeof_x` = 23

FPC compiler internal procedure index: `typeof`.

`fpc_in_unbox_x_y` = 78

FPC compiler internal procedure index: `unbox`.

`fpc_in_unpack_x_y_z` = 60

FPC compiler internal procedure index: unpack.

```
fpc_in_val_x = 24
```

FPC compiler internal procedure index: val.

```
fpc_in_writeln_x = 15
```

FPC compiler internal procedure index: writeln.

```
fpc_in_writestr_x = 62
```

Internal write string procedure.

```
fpc_in_write_x = 14
```

FPC compiler internal procedure index: write.

```
fpc_in_x86_cli = fpc_in_cpu_first + 6
```

```
fpc_in_x86_get_cs = fpc_in_cpu_first + 8
```

```
fpc_in_x86_get_ds = fpc_in_cpu_first + 10
```

```
fpc_in_x86_get_es = fpc_in_cpu_first + 11
```

```
fpc_in_x86_get_fs = fpc_in_cpu_first + 12
```

```
fpc_in_x86_get_gs = fpc_in_cpu_first + 13
```

```
fpc_in_x86_get_ss = fpc_in_cpu_first + 9
```

```
fpc_in_x86_inportb = fpc_in_cpu_first
```

```
fpc_in_x86_inportl = fpc_in_cpu_first + 2
```

```
fpc_in_x86_inportw = fpc_in_cpu_first + 1
```

```
fpc_in_x86_outportb = fpc_in_cpu_first + 3
```

```
fpc_in_x86_outportl = fpc_in_cpu_first + 5
```

`fpc_in_x86_outportw = fpc_in_cpu_first + 4`

`fpc_in_x86_sti = fpc_in_cpu_first + 7`

`fpc_in_xor_assign_x_y = 88`

`fpc_objc_encode_x = 71`

`fpc_objc_protocol_x = 70`

`fpc_objc_selector_x = 69`

`growheapsize1 : PtrUInt = 256 * 1024`

Grow rate for block less than 256 Kb.

`growheapsize2 : PtrUInt = 1024 * 1024`

Grow rate for block larger than 256 Kb.

`growheapsize_small : PtrUInt = 32 * 1024`

Fixed size small blocks grow rate.

`has_mmx_support : Boolean = False`

CPU has mmx support.

`has_sse2_support : Boolean = False`

CPU has sse2 support?

`has_sse3_support : Boolean = False`

CPU has sse3 support?

`has_sse_support : Boolean = False`

CPU has SSE support ?

`InitProc : CodePointer = Nil`

`InitProc` is a routine that can be called after all units were initialized. It can be set by units to execute code that can be initialized after all units were initialized.

Remark When setting the value of `InitProc`, the previous value should always be saved, and called when the installed initialization routine has finished executing.

```
IOBJECTINSTANCE : TGUID =
  '{D91C9AF4-3C93-420F-A303-BF5BA82BFD23}'
```

`IOBJECTINSTANCE` is an internal GUID, which should not be used in end-user code. It is used in the `as` operator.

```
ISMULTITHREAD : longbool = False
```

Indicates whether more than one thread is running in the application.

```
LFNSUPPORT = True
```

`LFNSUPPORT` determines whether the current OS supports long file names, i.e. filenames that are not of the form 8.3 as on ancient DOS systems. If the value of this constant is `True` then long filenames are supported. If it is false, then not.

This constant is part of a set of constants that describe the OS characteristics. These constants should be used instead of hardcoding OS characteristics.

```
LINEENDING = #10
```

`LINEENDING` is a constant which contains the current line-ending character. This character is system dependent, and is initialized by the system. It should not be set.

This constant is part of a set of constants that describe the OS characteristics. These constants should be used instead of hardcoding OS characteristics.

```
MAXEXITCODE = 255
```

`MAXEXITCODE` is the maximum value for the `Halt` ([1481](#)) call.

```
MAXINT = maxsmallint
```

Maximum integer value.

```
MAXKEPTOSCHUNKS : DWord = 4
```

`MAXKEPTOSCHUNKS` tells the heap manager how many free chunks of OS-allocated memory it should keep in memory. When freeing memory, it can happen that a memory block obtained from the OS is completely free. If more than `MAXKEPTOSCHUNKS` such blocks are free, then the heap manager will return them to the OS, to reduce memory requirements.

```
MAXLONGINT = $7fffffff
```

Maximum longint value.

```
MAXPATHLEN = 4096
```

This constant is system dependent.

```
MAXSINTVALUE = High(ValSInt)
```

Maximum String-size value.

`MaxSmallint = 32767`

Maximum smallint value.

`MaxUIntValue = High(ValUInt)`

Maximum unsigned integer value.

`Max_Frame_Dump : Word = 8`

Maximum number of frames to show in error frame dump.

`ModuleIsCpp : Boolean = False`

`ModuleIsCpp` is always false for FPC programs, it is provided for Delphi compatibility only.

`ModuleIsLib : Boolean = False`

`ModuleIsLib` is set by the compiler when linking a library, program or package, and determines whether the current module is a library (or package) (`True`) or program (`False`).

`ModuleIsPackage : Boolean = False`

`ModuleIsLib` is set by the compiler when linking a library, program or package, and determines whether the current module is a package (`True`) or a library or program (`False`).

`NilHandle = TLibHandle(0)`

Correctly typed Nil handle - returned on error by `LoadLibrary` ([1503](#)).

`PathSeparator = ':'`

`PathSeparator` is the character used commonly on the current operating system to separate paths in a list of paths, such as the `PATH` environment variable.

This constant is part of a set of constants that describe the OS characteristics. These constants should be used instead of hardcoding OS characteristics.

`PtrToNil : Pointer = Nil`

`RaiseMaxFrameCount : LongInt = 16`

Maximum number of frames to include in `TExceptObject` ([1389](#)).

`RaiseProc : TExceptProc = Nil`

Procedure to raise an exception.

`RT_ACCELERATOR = MAKEINTRESOURCE(9)`

Constant identifying an accelerator resource.

RT_ANICURSOR = MAKEINTRESOURCE (21)

This constant can be used to specify a resource of type "animated cursor".

RT_ANIICON = MAKEINTRESOURCE (22)

This constant can be used to specify a resource of type "animated icon".

RT_BITMAP = MAKEINTRESOURCE (2)

Constant identifying a bitmap resource.

RT_CURSOR = MAKEINTRESOURCE (1)

Constant identifying a cursor resource.

RT_DIALOG = MAKEINTRESOURCE (5)

Constant identifying a dialog resource.

RT_FONT = MAKEINTRESOURCE (8)

Constant identifying a font resource.

RT_FONTDIR = MAKEINTRESOURCE (7)

Constant identifying a font directory resource.

RT_GROUP_CURSOR = MAKEINTRESOURCE (12)

Constant identifying a group cursor resource.

RT_GROUP_ICON = MAKEINTRESOURCE (14)

Constant identifying a group icon resource.

RT_HTML = MAKEINTRESOURCE (23)

This constant can be used to specify a resource of type "HTML data".

RT_ICON = MAKEINTRESOURCE (3)

Constant identifying an icon resource.

RT_MANIFEST = MAKEINTRESOURCE (24)

This constant can be used to specify a resource of type "Manifest".

RT_MENU = MAKEINTRESOURCE (4)

Constant identifying a menu resource.

```
RT_MESSAGETABLE = MAKEINTRESOURCE(11)
```

Constant identifying a message data resource.

```
RT_RCDATA = MAKEINTRESOURCE(10)
```

Constant identifying a binary data resource.

```
RT_STRING = MAKEINTRESOURCE(6)
```

Constant identifying a string table resource.

```
RT_VERSION = MAKEINTRESOURCE(16)
```

Constant identifying a version info resource.

```
RuntimeErrorExitCodes : Array[TRuntimeError] of Byte = (0, 203, 204
, 200, 201, 215, 207, 200, 205, 206, 219, 216, 218, 217, 202, 220
, 221, 222, 223, 224, 225, 227, 212, 228, 229, 233, 234, 235, 236
)
```

This array is used by the [Error \(1459\)](#) routine to convert a [TRuntimeError \(1397\)](#) enumeration type to a process exit code.

```
SafeCallErrorProc : TSafeCallErrorProc = Nil
```

`SafeCallErrorProc` is a Handler called in case of a safecall calling convention error. `Error` is the error number (passed by the Windows operating system) and `Addr` is the address where the error occurred.

```
SharedSuffix = 'so'
```

Shared library suffix for the current platform.

```
sLineBreak = LineEnding
```

`sLineBreak` is an alias for [LineEnding \(1358\)](#) and is supplied for Delphi compatibility.

This constant is part of a set of constants that describe the OS characteristics. These constants should be used instead of hardcoding OS characteristics.

```
StackError : Boolean = False
```

Indicate whether there was a stack error.

```
StdErrorHandle = 2
```

Value of the OS handle for the standard error-output file.

```
StdInputHandle = 0
```

Value of the OS handle for the standard input file.

`StdOutputHandle = 1`

Value of the OS handle for the standard output file.

`S_FALSE = 1`

Interface call result: Not OK.

`S_OK = 0`

Interface call result: OK.

`Test8086 : Byte = 2`

This constant will only exist on Intel CPU platforms.

`Test8087 : Byte = 3`

This constant will only exist on Intel CPU platforms.

`TextRecBufSize = 256`

`TextRecBufSize` is the default buffer size for text files. The actual buffer can be set to another size using `SetTextBuf` ([1542](#)).

`TextRecNameLength = 256`

`TextRecNameLength` is the maximum filename size for text files.

`ThreadingAlreadyUsed : Boolean = False`

Internal constant for the threading system. Don't use.

`tkAnsiChar = tkChar`

`tkAnsiChar` is an alias for `TTypeKind.tkChar` ([1401](#))

`tkAnsiString = tkAString`

`tkAnsiString` is an alias for `TTypeKind.tkAString` ([1401](#))

`tkShortString = tkSString`

`tkShortString` is an alias for `TTypeKind.tkSString` ([1401](#))

`tkUnicodeString = tkUString`

`tkUnicodeString` is an alias for `TTypeKind.tkUString` ([1401](#))

`tkWideChar = tkWChar`

tkWideChar is an alias for TTypeKind.tkUChar ([1401](#))

tkWideString = tkWString

tkWideString is an alias for TTypeKind.tkWString ([1401](#))

```
UnixGetModuleByAddrHook : procedure(addr: pointer; var baseaddr: pointer
    ; var filename: string) = Nil
```

UnixGetModuleByAddrHook is used on Unix systems to retrieve a module name based on an address. It is used in the exeinfo ([763](#)) unit to map addresses to module (programs or library) names.

UnusedHandle = - 1

Value indicating an unused file handle (as reported by the OS).

```
VarAddRefProc : procedure(var v: tvardata) = Nil
```

Callback to increase reference count of a variant.

varAny = \$101

Variant type: Any.

varArray = \$2000

Variant type: variant Array.

varBoolean = 11

Variant type: Boolean type.

varByRef = \$4000

Variant type: By reference.

varByte = 17

Variant type: Byte (8 bit).

```
VarClearProc : procedure(var v: tvardata) = Nil
```

Callback to clear a variant.

```
VarCopyProc : procedure(var d: tvardata; const s: tvardata) = Nil
```

Callback to copy a variant.

varCurrency = 6

Variant type: Currency.

`varDate = 7`

Variant type: Date.

`varDecimal = 14`

Variant type: Decimal (BCD).

`varDispatch = 9`

Variant type: dispatch interface.

`varDouble = 5`

Variant type: Double float.

`varEmpty = 0`

Variant type: Empty variant.

`varError = 10`

Variant type: Error type.

`varInt64 = 20`

Variant type: Integer (64-Bit).

`varInteger = 3`

Variant type: Integer (32-bit).

`varLongWord = 19`

Variant type: Word (32 bit).

`varNull = 1`

Variant type: Null ([1510](#)) variant.

`varOleStr = 8`

Variant type: OLE string (widerstring).

`varQWord = 21`

Variant type: Word (64-bit).

`varRecord = 36`

Record variant type.

`varShortInt = 16`

Variant type: Shortint (16 bit).

`varSingle = 4`

Variant type: Single float.

`varSmallInt = 2`

Variant type: smallint (8 bit).

`varStrArg = $48`

Variant type: String.

`varString = $100`

Variant type: String.

`VarToLStrProc : procedure(var d: AnsiString; const s: tvardata) =
Nil`

Callback to convert a variant to a ansistring.

`VarToWStrProc : procedure(var d: WideString; const s: tvardata) =
Nil`

Callback to convert a variant to a widestring.

`varTypeMask = $fff`

Variant type: Mask to extract type.

`varUInt64 = varQWord`

`varuint64` denotes an unsigned 64-bit value in a variant. It is one of the values found in the `VType` field of the variant record `tvdata` ([1407](#)).

`varUnknown = 13`

Variant type: Unknown.

`varUStrArg = $49`

`varustrarg` denotes a Unicode string argument in `DispInvoke` call. It will be converted to `varustring` in a variant.

`varUString = $102`

`varustring` denotes a Unicode string value in a variant. It is one of the values found in the `VType` field of the variant record `tvdata` ([1407](#)).

`varVariant = 12`

Variant type: Variant (arrays only).

`varWord = 18`

Variant type: Word (16 bit).

`varWord64 = varQWord`

Variant type: Word (64-bit).

`vmtAfterConstruction = vmtMethodStart + sizeof(codepointer) * 5`

VMt Layout: ?

`vmtAutoTable = vmtParent + sizeof(pointer) * 7`

VMt layout: ?

`vmtBeforeDestruction = vmtMethodStart + sizeof(codepointer) * 6`

VMt Layout: ?

`vmtClassName = vmtParent + sizeof(pointer)`

VMt Layout: location of class name.

`vmtDefaultHandler = vmtMethodStart + sizeof(codepointer) * 4`

VMt Layout: ?

`vmtDefaultHandlerStr = vmtMethodStart + sizeof(codepointer) * 7`

VMt Layout: ?

`vmtDestroy = vmtMethodStart`

VMt Layout: Location of destructor pointer.

`vmtDispatch = vmtMethodStart + sizeof(codepointer) * 8`

`vmtDispatch` is the offset from the VMT start, in bytes to the dispatch table for a class. The dispatch table is used when dispatching messages in `TObject.Dispatch` ([1604](#))

`vmtDispatchStr = vmtMethodStart + sizeof(codepointer) * 9`

`vmtDispatchStr` is the offset from the VMT start, in bytes to the dispatch table for a class. The dispatch table is used when dispatching messages in `TObject.DispatchStr` ([1605](#))

`vmtDynamicTable = vmtParent + sizeof(pointer) * 2`

VMT Layout: location of dynamic methods table.

```
vmtEquals = vmtMethodStart + sizeof(codepointer) * 10
```

vmtEquals contains the offset from the VMT start, of the location of the TObjct.Equals (1607) method pointer.

```
vmtFieldTable = vmtParent + sizeof(pointer) * 4
```

VMT Layout: Location of fields table.

```
vmtFreeInstance = vmtMethodStart + sizeof(codepointer) * 2
```

VMT Layout: location of FreeInstance method.

```
vmtGetHashCode = vmtMethodStart + sizeof(codepointer) * 11
```

vmtGetHashCode contains the offset from the VMT start, of the location of the TObjct.GetHashCode (1607) method pointer.

```
vmtInitTable = vmtParent + sizeof(pointer) * 6
```

VMT Layout: ?

```
vmtInstanceSize = 0
```

VMT Layout: Location of class instance size in VMT.

```
vmtIntfTable = vmtParent + sizeof(pointer) * 8
```

VMT layout: Interface table.

```
vmtMethodStart = vmtParent + sizeof(pointer) * 10
```

VMT layout: start of method table.

```
vmtMethodTable = vmtParent + sizeof(pointer) * 3
```

VMT Layout: Method table start.

```
vmtMsgStrPtr = vmtParent + sizeof(pointer) * 9
```

VMT layout: message strings table.

```
vmtNewInstance = vmtMethodStart + sizeof(codepointer)
```

VMT Layout: location of NewInstance method.

```
vmtParent = sizeof(SizeInt) * 2
```

VMT Layout: location of pointer to parent VMT.

`vmtSafeCallException = vmtMethodStart + sizeof(codepointer) * 3`

VMT Layout: ?

`vmtToString = vmtMethodStart + sizeof(codepointer) * 12`

`vmtToString` contains the offset from the VMT start, of the location of the `TObject.ToString` (1608) method pointer.

`vmtTypeInfo = vmtParent + sizeof(pointer) * 5`

VMT Layout: Location of class type information.

`vtAnsiString = 11`

TVarRec type: Ansistring.

`vtBoolean = 1`

TVarRec type: Boolean.

`vtChar = 2`

TVarRec type: Char.

`vtClass = 8`

TVarRec type: Class type.

`vtCurrency = 12`

TVarRec type: Currency.

`vtExtended = 3`

TVarRec type: Extended.

`vtInt64 = 16`

TVarRec type: Int64 (signed 64-bit integer).

`vtInteger = 0`

TVarRec type: Integer.

`vtInterface = 14`

TVarRec type: Interface.

`vtObject = 7`

TVarRec type: Object instance.

vtPChar = 6

TVarRec type: PChar.

vtPointer = 5

TVarRec type: pointer.

vtPWideChar = 10

TVarRec type: PWideChar.

vtQWord = 17

TVarRec type: QWord (unsigned 64-bit integer).

vtString = 4

TVarRec type: String.

vtUnicodeString = 18

vtUnicodeString denotes a Unicode string argument in the array of const. The TVarRec.VUnicodeString field will contain the actual value.

vtVariant = 13

TVarRec type: Variant.

vtWideChar = 9

TVarRec type: Widechar.

vtWideString = 15

TVarRec type: WideString.

75.11.2 Types

AnsiChar = Char

Alias for 1-byte sized char.

Byte = 0..255

An unsigned 8-bits integer.

Cardinal = LongWord

An unsigned 32-bits integer.

`Char = #0..#255`

`Char` is the basic ANSI character type, with a range of `[#0..#255]`.

`CodePointer = Pointer`

`CodePointer` is used in 8/16-bit targets to indicate pointers to code segments. On all other platforms this equals `Pointer`.

`CodePointer = Pointer`

`CodePointer` is used in 8/16-bit targets to indicate pointers to code segments. On all other platforms this equals `Pointer`.

`CodePtrInt = PtrInt`

`CodePtrInt` is a signed integer with the same size as `CodePointer` ([1370](#))

`CodePtrInt = PtrInt`

`CodePtrInt` is a signed integer with the same size as `CodePointer` ([1370](#))

`CodePtrUInt = PtrUInt`

`CodePtrUInt` is an unsigned integer with the same size as `CodePointer` ([1370](#))

`CodePtrUInt = PtrUInt`

`CodePtrUInt` is an unsigned integer with the same size as `CodePointer` ([1370](#))

`Comp = Int64`

`Comp` is a 64-bit floating point type. See the language reference for more details.

`DWord = LongWord`

An unsigned 32-bits integer.

```
EnumResLangProc = function(ModuleHandle: TFPResourceHMODULE;
    ResourceType: PChar; ResourceName: PChar;
    IDLanguage: Word; lParam: PtrInt) :
    LongBool
```

`EnumResNameProcs` used in the `EnumResourceLanguages` ([1455](#)) call. It is called for all languages for a resource of the specified type and name, and is passed the `ModuleHandle`, `ResourceName`, `ResourceName` and `IDLanguage` values for each language encountered for the specified resource. Additionally, the `lParam` parameter from the `EnumResourceLanguages` is passed unaltered.

```
EnumResNameProc = function(ModuleHandle: TFPResourceHMODULE;
    ResourceType: PChar; ResourceName: PChar;
    lParam: PtrInt) : LongBool
```

EnumResNameProc is used in the EnumResourceNames (1456) call. It is called for all resources of the specified type, and is passed the ModuleHandle, ResourceType, ResourceName values for each resource encountered. Additionally, the lParam parameter from the EnumResourceNames is passed unaltered.

```
EnumResTypeProc = function(ModuleHandle: TFPResourceHMODULE;
    ResourceType: PChar; lParam: PtrInt)
    : LongBool
```

EnumResTypeProc is used in the EnumResourceTypes (1456) call. It is called for all resources, and is passed the ModuleHandle, ResourceType values for each resource encountered. Additionally, the lParam parameter from the EnumResourceTypes is passed unaltered.

```
FarPointer = Pointer
```

FarPointer is used in 8/16-bit targets to indicate far pointers (over segments). On all other platforms this equals Pointer.

```
FileRec = record
public
    Handle : THandle;
    Mode : LongInt;
    RecSize
    : SizeInt;
    _private : Array[1..3*SizeOf(SizeInt)+5*SizeOf(pointer
    )] of Byte;
    UserData : Array[1..32] of Byte;
    name : Array[0..filerecnamelength
    ] of TFileTextRecChar;
end
```

FileRec is the underlying type used in untyped files. It should be treated as opaque and never manipulated directly.

```
HGLOBAL = PtrUInt
```

This is an opaque type.

```
HMODULE = PtrUInt
```

This is an opaque type.

```
HRESULT = LongInt
```

32-Bit signed integer.

```
IInterface = IUnknown
```

`IInterface` is the basic interface from which all COM style interfaces descend.

`Int16 = SmallInt`

A signed 16-bits integer.

`Int32 = LongInt`

A signed 32-bits integer.

`Int64 = - 9223372036854775808..9223372036854775807`

`Int64` is a 64-bit, signed integer type, with range [-9223372036854775808..9223372036854775807].

`Int8 = ShortInt`

A signed 8-bits integer.

`Integer = SmallInt`

The system unit defines `Integer` as a signed 16-bit integer. But when DELPHI or OBJFPC mode are active, then the `objpas` unit redefines `Integer` as a 32-bit integer.

Remark Note that due to the way the language modes are implemented, if you refer to `integer` with the fully qualified name `System.Integer`, you will still get a 16-bit integer.

`IntegerArray = Array[0..$ffffff] of Integer`

Generic array of integer.

`IntPtr = PtrInt`

A signed integer with the same size in bytes as pointer.

```
jmp_buf = packed record
public
    ebx : LongInt;
    esi : LongInt;
    edi : LongInt;
    bp  : Pointer;
    sp  : Pointer;
    pc  : Pointer;
end
```

Record type to store processor information.

`Longint = - 2147483648..2147483647`

A signed 32-bits integer.

`Longword = 0..4294967295`

The base 32-bit unsigned type. See the reference manual for more details

`MAKEINTRESOURCE = PChar`

Alias for the `PChar` (1374) type.

`MarshaledAString = PAnsiChar`

Alias for `PAnsiChar`.

`MarshaledString = PWideChar`

Alias for `PWideChar`.

`NativeInt = PtrInt`

`NativeInt` is defined for Delphi compatibility. It is a signed integer with the size of a pointer, so 32-bit on 32-bit platforms, 64-bit on 64-bit platforms. It is advisable to use unsigned variants of this type. See `PtrUInt` (1381) for details.

`NativeUInt = PtrUInt`

`NativeUInt` is defined for Delphi compatibility. It is an unsigned integer with the size of a pointer, so 32-bit on 32-bit platforms, 64-bit on 64-bit platforms.

`OpaquePointer = POpaqueData`

Type alias for `POpaqueData`.

`PAnsiChar = PChar`

Alias for `PChar` (1374) type.

`PAnsiString = ^AnsiString`

Pointer to an `ansistring` type.

`PBoolean = ^Boolean`

Pointer to a `Boolean` type.

`PBoolean16 = ^Boolean16`

Pointer to `Boolean16` type.

`PBoolean32 = ^Boolean32`

Pointer to `Boolean32` type.

`PBoolean64 = ^Boolean64`

Pointer to Boolean64 type.

```
PBoolean8 = ^Boolean8
```

Pointer to Boolean8 type.

```
PByte = ^Byte
```

Pointer to byte (1369) type.

```
PByteBool = ^ByteBool
```

Pointer to ByteBool type.

```
pcallldesc = ^tcallldesc
```

Pointer to TCallDesc (1385) record.

```
PCardinal = ^Cardinal
```

Pointer to Cardinal (1370) type.

```
PChar = ^Char
```

Or the same as a pointer to an array of char. See the reference manual for more information about this type.

```
PClass = ^TClass
```

Pointer to TClass (1385).

```
PCodePointer = ^CodePointer
```

PCodePointer is a typed pointer to CodePointer (1370).

```
PCurrency = ^Currency
```

Pointer to currency type.

```
PDate = ^TDateTime
```

Pointer to a TDateTime (1386) type.

```
PDateTime = ^TDateTime
```

Pointer to TDateTime.

```
PDispatch = ^IDispatch
```

Pointer to IDispatch (1590) interface type.

`pdispdesc = ^tdispdesc`

Pointer to `tdispdesc` (1387) record.

`PDouble = ^Double`

Pointer to double-sized float value.

`PDWord = ^DWord`

Pointer to `DWord` (1370) type.

`pdynarrayindex = ^tdynarrayindex`

Pointer to `tdynarrayindex` (1387) type.

`pdynarraytypeinfo = ^tdynarraytypeinfo`

Pointer to `TDynArrayTypeInfo` (1387) type.

`PError = ^TError`

Pointer to an `Error` (1459) type.

`PEventState = pointer`

Pointer to `EventState`, which is an opaque type.

`PExceptAddr = ^TExceptAddr`

`PExceptAddr` is a pointer to `PExceptAddr` (1375)

`PExceptObject = ^TExceptObject`

Pointer to Exception handler procedural type `TExceptProc` (1389).

`PExtended = ^Extended`

Pointer to extended-sized float value.

`PFileTextRecChar = ^TFileTextRecChar`

`PFileTextRecChar` is a typed pointer to `TFileTextRecChar` (1390).

`PGuid = ^TGuid`

Pointer to `TGUID` (1583) type.

`PInt16 = PSmallInt`

Pointer to `Int16` (1372) type.

`PInt32 = PLongint`

Pointer to Int32 (1372) type.

`PInt64 = ^Int64`

Pointer to Int64 type.

`PInt8 = PShortInt`

Pointer to Int8 (1372) type.

`PInteger = ^Integer`

Pointer to integer (1372) type.

`PIntegerArray = ^IntegerArray`

Pointer to IntegerArray (1372) type.

`PInterface = PUnknown`

Pointer to IInterface (1372) interface.

`pinterfaceentry = ^tinterfaceentry`

Pointer to tinterfaceentry (1585) record.

`pinterfacetable = ^tinterfacetable`

Pointer to tinterfacetable (1393) record.

`PIntPtr = PPtrInt`

Pointer to IntPtr (1372) type.

`PJump_buf = ^jmp_buf`

Pointer to jmp_buf (1372) record.

`PLongBool = ^LongBool`

Pointer to a LongBool type.

`PLongint = ^LongInt`

Pointer to Longint (1372) type.

`PLongWord = ^LongWord`

Pointer to LongWord type.

`PMarshaledAString = ^PAnsiChar`

Pointer to `PWideChar`.

`PMarshaledString = ^PWideChar`

Pointer to `PWideChar`.

`PMemoryManager = ^TMemoryManager`

Pointer to `TMemoryManager` (1394) record.

`PMethod = ^TMethod`

Pointer to method.

`PMsgStrTable = ^TMsgStrTable`

Pointer to array of `TMsgStrTable` (1394) records.

`PNativeInt = ^NativeInt`

Pointer to `NativeInt` (1373) type.

`PNativeUInt = ^NativeUInt`

Pointer to `NativeInt` (1373) type.

`PointerArray = Array[0..512*1024*1024-2] of Pointer`

Generic pointer array.

`PoleVariant = ^OleVariant`

Pointer to `OleVariant` type.

`POpaqueData = ^TOpaqueData`

`POpaqueData` represents a pointer to data for which the internal structure must not be known. You can make type aliases from this pointer.

`PPAnsiChar = PPChar`

Alias for `PPChar` (1378) type.

`PPByte = ^PByte`

`PPByte` is a pointer to a `PByte` (1374) type.

`PPChar = ^PChar`

Pointer to an array of pointers to null-terminated strings.

`PPCharArray = ^TPCharArray`

Pointer to `TPCharArray` (1395) type.

`PPCodePointer = ^PCodePointer`

`PPCodePointer` is a typed pointer to `PCodePointer` (1374).

`PPDispatch = ^PDispatch`

Pointer to `PDispatch` (1374) pointer type.

`PPDouble = ^PDouble`

`PPDouble` is a pointer to a `PDouble` (1375) type.

`ppdynarraytypeinfo = ^pdynarraytypeinfo`

Pointer to `pdynarraytypeinfo`.

`PPLongint = ^PLongint`

`PPLongint` is a pointer to a `PLongint` (1376) type.

`PPPointer = ^Pointer`

Pointer to a pointer type.

`PPPointerArray = ^PointerArray`

Pointer to `PointerArray` (1377) type.

`PPPAansiChar = PPPChar`

`PPPAansiChar` is a typed pointer to `PPAnsichar` (1377).

`PPPChar = ^PPChar`

`PPPChar` is a pointer to a `PPChar` (1378)

`PPPointer = ^PPointer`

Pointer to a `PPointer` (1378) type.

`PPPWideChar = ^PPWideChar`

`PPPWideChar` is a pointer to a `PPWideChar` (1379) type.

`PPPtrInt = ^PtrInt`

Pointer to `PtrInt` (1380) type.

```
PPtrUInt = ^PtrUInt
```

Pointer to unsigned integer of pointer size.

```
PPUnknown = ^PUnknown
```

Pointer to untyped pointer.

```
PPVmt = ^PVmt
```

Pointer to `PVMT` pointer.

```
PPWideChar = ^PWideChar
```

Pointer to link id="`PWideChar`"> type.

```
PQWord = ^QWord
```

Pointer to `QWord` type.

```
PQWordBool = ^QWordBool
```

Pointer to `QWordBool` type.

```
PRawByteString = ^RawByteString
```

Pointer to `RawByteString`.

```
PRTLCriticalSection = ^RTLCriticalSection
```

Pointer to `#rtl.system.RTLCriticalSection` (1396) type.

```
PRTLEvent = pointer
```

Pointer to `RTLEvent`, which is an opaque type.

```
PShortInt = ^ShortInt
```

Pointer to `shortint` (1383) type.

```
PShortString = ^ShortString
```

Pointer to a `shortstring` type.

```
PSingle = ^Single
```

Pointer to single-sized float value.

```
PSizeInt = ^SizeInt
```

Pointer to a `SizeInt` (1383) type.

```
PSizeUInt = ^SizeUInt
```

`PSizeUInt64` is a pointer to a `SizeUInt` (1384), an unsigned integer of architecture-dependent size.

```
PSmallInt = ^SmallInt
```

Pointer to `smallint` (1384) type.

```
pstringmessagetable = ^TStringMessageTable
```

Pointer to `TStringMessageTable` (1398) record.

```
PText = ^Text
```

Pointer to text file.

```
PtrInt = Int64
```

`PtInt` is a signed integer type which has always the same size as a pointer. `PtInt` is considered harmful and should almost never be used in actual code, because pointers are normally unsigned. For example, consider the following code:

```
getmem(p, 2048);           {Assume the address of p becomes $7fffffff0.}
q:=pointer(pInt(p)+1024);  {Overflow error.}
writeln(q>p);             {Incorrect answer.}
```

`PtInt` might have a valid use when two pointers are subtracted from each other if it is unknown which pointer has the largest address. However, even in this case `pInt` causes trouble in case the distance is larger than `high(pInt)` and must be used with great care.

The introduction of the `pInt` type was a mistake. Please use `PtrUInt` (1381) instead.

```
PtrInt = LongInt
```

`PtInt` is a signed integer type which has always the same size as a pointer. `PtInt` is considered harmful and should almost never be used in actual code, because pointers are normally unsigned. For example, consider the following code:

```
getmem(p, 2048);           {Assume the address of p becomes $7fffffff0.}
q:=pointer(pInt(p)+1024);  {Overflow error.}
writeln(q>p);             {Incorrect answer.}
```

`PtInt` might have a valid use when two pointers are subtracted from each other if it is unknown which pointer has the largest address. However, even in this case `pInt` causes trouble in case the distance is larger than `high(pInt)` and must be used with great care.

The introduction of the `pInt` type was a mistake. Please use `PtrUInt` (1381) instead.

```
PtrUInt = DWord
```

`PtrUInt` is an unsigned integer type which has always the same size as a pointer. When using integers which will be cast to pointers and vice versa, use this type, never the regular `Cardinal` type.

`PtrUInt = QWord`

`PtrUInt` is an unsigned integer type which has always the same size as a pointer. When using integers which will be cast to pointers and vice versa, use this type, never the regular `Cardinal` type.

`PUCS2Char = PWideChar`

Pointer to `UCS2Char` (1412) character.

`PUCS4Char = ^UCS4Char`

Pointer to `UCS4Char` (1412).

`PUCS4CharArray = ^TUCS4CharArray`

Pointer to array of `UCS4Char` (1412) characters.

`PUInt16 = PWord`

Pointer to `UInt16` (1412) type.

`PUInt32 = PDWord`

Pointer to `UInt32` (1412) type.

`PUInt64 = ^UInt64`

`PUInt64` is a pointer to a `UInt64` (1412), an unsigned 64-bit integer.

`PUInt8 = PByte`

Pointer to `UInt8` (1412) type.

`PUIntPtr = PPtrUInt`

Pointer to `UIntPtr` (1412) type.

`PUnicodeChar = ^UnicodeChar`

`PUnicodeChar` is a pointer to a Unicode character, just like `PChar` is a pointer to a Char an-sistring character.

`PUnicodeString = ^UnicodeString`

`PUnicodeString` is a pointer to a `UnicodeString` string.

`PUnknown = ^IUnknown`

Untyped pointer.

`PUTF8Char = PAnsiChar`

Pointer to UTF8Char.

```
PUTF8String = ^UTF8String
```

Pointer to UTF8String (1413).

```
pvararray = ^tvararray
```

Pointer to TVarArray (1404) type.

```
pvararraybound = ^tvararraybound
```

Pointer to tvararraybound (1404) type.

```
pvararrayboundarray = ^tvararrayboundarray
```

Pointer to tvararrayboundarray (1404) type.

```
pvararraycoorarray = ^tvararraycoorarray
```

Pointer to tvararraycoorarray (1404) type.

```
pvardata = ^tvardata
```

Pointer to TVarData (1407) record.

```
PVariant = ^Variant
```

Pointer to Variant type.

```
pvariantmanager = ^tvariantmanager
```

Pointer to TVariantManager (1409) record.

```
PVarRec = ^TVarRec
```

Pointer to TVarRec (1411) type.

```
PVmt = ^TVmt
```

Pointer to TVMT (1589) record.

```
PWideChar = ^WideChar
```

Pointer to WChar (1413).

```
PWideString = ^WideString
```

Pointer to widestring type.

```
PWord = ^Word
```

Pointer to word (1413) type.

```
PWordBool = ^WordBool
```

Pointer to a WordBool type.

```
QWord = 0..18446744073709551615
```

QWord is a 64-bit, unsigned integer type, with range [0..18446744073709551615].

```
RawByteString = ansistring
```

RawByteString is a single-byte character string which does not have any codepage associated with it.

This means that assigning a single-byte character string to this kind of string will not change the codepage of the string.

Inversely, when assigning a RawByteString to a single-byte string, the codepage of the destination is simply set to the codepage of the rawbytestring: no codepage conversion happens, the reference count is simply increased.

```
Real = Double
```

Alias for real type.

```
Real48 = Array[0..5] of Byte
```

TP compatible real type (6 bytes) definition.

```
Shortint = - 128..127
```

A signed 8-bits integer.

```
SizeInt = Int64
```

SizeInt is used to describe sizes of structures in FPC using a signed integer. It is an opaque type: The actual type of this type depends on the architecture: its size reflects the maximum addressable memory on the current architecture, thus it is 64-bit on 64-bit platforms, 32-bit on 32-bit platforms, and 16 bit on 16 bit platforms.

If you wish to create cross-platform code, do not use the actual type of SizeInt displayed here, it reflects the definition used on the platform used to generate the documentation.

```
SizeInt = LongInt
```

SizeInt is used to describe sizes of structures in FPC using a signed integer. It is an opaque type: The actual type of this type depends on the architecture: its size reflects the maximum addressable memory on the current architecture, thus it is 64-bit on 64-bit platforms, 32-bit on 32-bit platforms, and 16 bit on 16 bit platforms.

If you wish to create cross-platform code, do not use the actual type of SizeInt displayed here, it reflects the definition used on the platform used to generate the documentation.

`SizeUInt = QWord`

`SizeUInt` is used to describe sizes of structures in FPC using an unsigned integer. The actual type of this type depends on the architecture: its size reflects the maximum addressable memory on the current architecture, thus it is 64-bit on 64-bit platforms, 32-bit on 32-bit platforms, and 16 bit on 16 bit platforms.

`SizeUInt = DWord`

`SizeUInt` is used to describe sizes of structures in FPC using an unsigned integer. The actual type of this type depends on the architecture: its size reflects the maximum addressable memory on the current architecture, thus it is 64-bit on 64-bit platforms, 32-bit on 32-bit platforms, and 16 bit on 16 bit platforms.

`Smallint = - 32768..32767`

A signed 16-bits integer.

`TAbstractErrorProc = procedure`

Abstract error handler procedural type.

`TAllocateThreadVarsHandler = procedure`

Threadvar allocation callback type for `TThreadManager` (1400).

`TAnsiChar = Char`

Alias for 1-byte sized char.

```
TAssertErrorProc = procedure(const msg: ShortString;
    const fname: ShortString; lineno: LongInt;
    erroraddr: pointer)
```

Assert error handler procedural type.

`TBacktraceStrFunc = function(Addr: CodePointer) : ShortString`

Type for formatting of backtrace dump.

```
TBasicEventCreateHandler = function(EventAttributes: Pointer;
    AManualReset: Boolean;
    InitialState: Boolean;
    const Name: ansistring)
    : PEventState
```

callback type for creating eventstate in `TThreadManager` (1400).

`TBasicEventHandler = procedure(state: PEventState)`

Generic callback type for handling eventstate in `TThreadManager` (1400).

```
TBasicEventWaitForHandler = function(timeout: Cardinal;
  state: PEventState;
  FUseComWait: Boolean) : LongInt
```

Wait for basic event callback type for TThreadManager (1400).

```
TBeginThreadHandler = function(sa: Pointer; stacksize: PtrUInt;
  ThreadFunction: TThreadFunc; p: pointer
  ;
                                creationFlags: DWord;
  var ThreadId: TThreadID) : TThreadID
```

Callback for thread start in TThreadManager (1400).

```
TBoundArray = Array of SizeInt
```

Dynamic array of integer.

```
tcalldesc = packed record
public
  calltype : Byte;
  argcount : Byte
  ;
  namedargcount : Byte;
  argtypes : Array[0..255] of Byte;
end
```

tcalldesc is used to encode the arguments to a dispatch call to an OLE dual interface. It is used on windows only. It describes the arguments to a call.

```
TClass = Class of TObject
```

Class of TObject (1597).

```
TCompareOption = (coIgnoreCase)
```

Table 75.10: Enumeration values for type TCompareOption

Value	Explanation
coIgnoreCase	Ignore case (usually identical to coLingIgnoreCase).

TCompareOption indicates how 2 strings should be compared. This option is used in the WideString-Manager (1412) implementation when comparing 2 strings. The following options exist:

Ignore case linguistically (usually identical to coIgnoreCase).

Ignore diacritic characters.

Ignore case (usually identical to coLingIgnoreCase).

Corresponding hiragana and katakana characters compare as equal.

Corresponding hiragana and katakana characters compare as equal.

Ignore nonspace characters (usually identical to `coLingIgnoreDiacritic`).

Ignore symbols and punctuation characters.

Ignore half-width and full width characters (used in Chinese and Japanese).

Use linguistic rules for casing, instead of file system rules.

Treat digits as numbers (20 before 120).

Handle punctuation as symbols.

Not all platforms will support all possibilities.

`TCompareOptions` = Set of `TCompareOption`

`TCompareOptions` is simply a set of `TCompareOption` (1385) enumeration values.

`TCriticalSectionHandler` = procedure(var cs)

Generic callback type for critical section handling in `TThreadManager` (1400).

`TCriticalSectionHandlerTryEnter` = function(var cs) : LongInt

`TCriticalSectionHandlerTryEnter` is the function prototype for the `TryEnterCriticalSection` (1563) function, in the `TThreadManager` (1400) record's `TryEnterCriticalSection` field.

`TCtrlBreakHandler` = function(CtrlBreak: Boolean) : Boolean

`TCtrlBreakHandler` is the prototype for the CTRL-C handler. If `CtrlBreak` is `True` then CTRL-Break was hit, otherwise CTRL-C was hit. The handlers should return `True` to signal that the key-combination was handled. If `False` is returned, then default handling will be used, which by default means an exception will be raised if the `sysutils` unit is used.

`TDate` = `TDateTime`

`TDate` is defined for Delphi compatibility. This type is deprecated, use `TDateTime` (1386) instead.

`TDateTime` = `Double`

Encoded Date-Time type.

```
tdispdesc = packed record
public
    dispid : LongInt;
    restype : Byte
    ;
    calldesc : tcalldesc;
end
```

`tcalldesc` is used to encode a dispatch call to an OLE dispatch interface. It is used on windows only. It describes the dispatch call call.

```
tdynarrayindex = SizeInt
```

A variable of type `tdynarrayindex` will always have the correct size, suitable for serving as an index in a dynamic array.

```
tdynarraytypeinfo = packed record
public
  kind : TTypeKind;
  namelen
  : Byte;
  elesize : SizeInt;
  eletype : ppdynarraytypeinfo;
  vartype
  : LongInt;
end
```

`tdynarraytypeinfo` describes the structure of a multi-dimensional dynamic array. It is used in the `DynArraySetLength` (1454) call.

```
TDynLibsManager = record
public
  LoadLibraryU : TLoadLibraryUHandler
  ;
  LoadLibraryA : TLoadLibraryAHandler;
  GetProcAddress : TGetProcAddressHandler
  ;
  GetProcAddressOrdinal : TGetProcAddressOrdinalHandler;
  UnloadLibrary
  : TUnloadLibraryHandler;
  GetLoadErrorStr : TGetLoadErrorStrHandler
  ;
end
```

`TDynLibsManager` contains all the callbacks needed to load and manage dynamic libraries. The system unit does not contain dynamic loading library support on all supported platforms. Like the unicode string support, heap support and thread support, support for loading dynamic libraries is pluggable. This record contains the necessary callbacks that the system unit needs to implement loading of dynamic libraries (needed for example for run-time package support).

Including the `dynlibs` (754) unit will enable support for dynamically loadable libraries on all platforms that support this.

```
TEndThreadHandler = procedure(ExitCode: DWord)
```

Callback for thread end in `TThreadManager` (1400).

```
TEntryInformation = record
public
  InitFinalTable : Pointer;
  ThreadvarTablesTable
```

```

: Pointer;
ResourceStringTables : Pointer;
ResStrInitTables
: Pointer;
ResLocation : Pointer;
PascalMain : procedure;
valgrind_used
: Boolean;
OS : TEntryInformationOS;
end

```

TEntryInformation is used to initialize a Free Pascal program or library. Under normal circumstances, there should be no need to use this structure directly: it is used by the system unit and special linking units.

```

TEntryInformationOS = record
public
  argc : LongInt;
  argv : PPChar
  ;
  envp : PPChar;
  stkptr : pointer;
  stklen : SizeUInt;
  haltproc
  : procedure(e: LongInt);
end

```

TEntryInformationOS represents executable entry information for the current OS. This structure is OS dependent.

```
TError = LongInt
```

Error type, used in variants.

```

TErrorProc = procedure(ErrNo: LongInt; Address: CodePointer;
  Frame: Pointer)

```

Standard error handler procedural type.

```

TExceptAddr = record
public
  buf : PJump_buf;
  next : PExceptAddr
  ;
  frametype : LongInt;
end

```

TExceptAddr describes an exception frame on the exception address frame stack. It is used in structured stack exception handling.

```

TExceptObject = record
public

```

```

FObject : TObject;
Addr : CodePointer
;
Next : PExceptObject;
refcount : LongInt;
Framecount : LongInt
;
Frames : PCodePointer;
end

```

TExceptObject is the exception description record which is found on the exception stack.

```

TExceptProc = procedure(Obj: TObject; Addr: CodePointer;
    FrameCount: LongInt; Frame: PCodePointer)

```

Exception handler procedural type.

```

TextBuf = Array[0..TextRecBufSize-1] of AnsiChar

```

TextBuf is a type for the default buffer used in TextRec ([1389](#)).

```

TextFile = Text

```

Alias for Text file type.

```

TextRec = record
public
    Handle : THandle;
    Mode : LongInt;
    bufsize
    : SizeInt;
    _private : SizeInt;
    bufpos : SizeInt;
    bufend : SizeInt
    ;
    bufptr : ^TextBuf;
    openfunc : CodePointer;
    inoutfunc : CodePointer
    ;
    flushfunc : CodePointer;
    closefunc : CodePointer;
    UserData
    : Array[1..32] of Byte;
    name : Array[0..textrecnamelength-1] of
    TFileTextRecChar;
    LineEnd : TLineEndStr;
    buffer : TextBuf;
end

```

TextRec is the underlying type used in text files. It should be treated as opaque and never manipulated directly.

TFileTextRecChar = UnicodeChar

TFileTextRecChar is the type of character used in TextRec (1389) or FileRec (1371) file types. It is an alias type, depending on platform and RTL compilation flags. No assumptions should be made on the actual character type.

```
TFloatSpecial = (fsZero, fsNZero, fsDenormal, fsNDenormal, fsPositive
,
                fsNegative, fsInf, fsNInf, fsNaN, fsInvalidOp)
```

Table 75.11: Enumeration values for type TFloatSpecial

Value	Explanation
fsDenormal	Denormal value.
fsInf	Infinity.
fsInvalidOp	Invalid operation.
fsNaN	Not a number.
fsNDenormal	Negative enormal value.
fsNegative	Negative value.
fsNInf	Negative infinity.
fsNZero	Negative zero.
fsPositive	Positive value.
fsZero	Zero.

TFloatSpecial enumerates a series of floating point value properties.

```
TFPCHeapStatus = record
public
    MaxHeapSize : PtrUInt;
    MaxHeapUsed
    : PtrUInt;
    CurrHeapSize : PtrUInt;
    CurrHeapUsed : PtrUInt;
    CurrHeapFree : PtrUInt;
end
```

TFPCHeapStatus describes the state of the FPC heap manager. This is not equivalent to the THeapStatus (1392) record defined by Delphi, which contains information not meaningful for the FPC heap manager. The heap status can be retrieved by the GetFPCHeapStatus (1474) call.

TFPResourceHandle = PtrUInt

TFPResourceHandle represents a handle to a binary resource and is used in the various resource calls. Its actual type and size may differ across platforms.

TFPResourceHGLOBAL = PtrUInt

TFPResourceHGLOBAL represents a handle to the global module containing a resource. It is used in the various resource calls. It is an opaque type: its actual type and size may differ across platforms.

TFPResourceHMODULE = PtrUInt

TFPResourceHMODULE represents a module (library, executable, other) in which a resource is located. It is used in the various resource calls. It is an opaque type: its actual type and size may differ across platforms.

```
TFPUException = (exInvalidOp, exDenormalized, exZeroDivide, exOverflow
,
exUnderflow, exPrecision)
```

Table 75.12: Enumeration values for type TFPUException

Value	Explanation
exDenormalized	Invalid denormalized form of float.
exInvalidOp	Invalid operation error.
exOverflow	Float overflow error.
exPrecision	Precision error.
exUnderflow	Float underflow error.
exZeroDivide	Division by zero error.

TFPUException describes what floating point errors raise exceptions. It has been moved here from the Math unit.

TFPUExceptionMask = Set of TFPUException

TFPUExceptionMask is a set of TFPUException constants

```
TFPUPrecisionMode = (pmSingle, pmReserved, pmDouble, pmExtended)
```

Table 75.13: Enumeration values for type TFPUPrecisionMode

Value	Explanation
pmDouble	Double-type precision.
pmExtended	Extended-type precision.
pmReserved	?
pmSingle	Single-type precision.

TFPUPrecisionMode describes the possible default precisions for the software Floating Point math routines. It has been moved here from the math unit.

```
TFPURoundingMode = (rmNearest, rmDown, rmUp, rmTruncate)
```

Table 75.14: Enumeration values for type TFPURoundingMode

Value	Explanation
rmDown	Round to biggest integer smaller than value.
rmNearest	Round to nearest integer.
rmTruncate	Cut off fractional part.
rmUp	Round to smallest integer larger than value.

TFPURoundingMode enumerates the possible values for software floating point math rounding. It has been moved here from the math unit.

```
TGetCurrentThreadIdHandler = function : TThreadId
```

Callback type for retrieving thread ID in TThreadManager (1400).

```
TGetLoadErrorStrHandler = function : string
```

TGetLoadErrorStrHandler is the type for the GetLoadErrorStr call using ansistring names in the TDynLibsManager (1387) dynamic library loading manager.

```
TGetProcAddressHandler = function(Lib: TLibHandle;
  const ProcName: AnsiString) : Pointer
```

TGetProcAddressHandler is the type for the GetProcAddress call using ansistring names in the TDynLibsManager (1387) dynamic library loading manager.

```
TGetProcAddressOrdinalHandler = function(Lib: TLibHandle;
  Ordinal: TOrdinalEntry)
  : Pointer
```

TGetProcAddressOrdinalHandler is the type for the GetProcAddressOrdinal call using ansistring names in the TDynLibsManager (1387) dynamic library loading manager.

```
THandle = LongInt
```

This type should be considered opaque. It is used to describe file and other handles.

```
THeapStatus = record
public
  TotalAddrSpace : Cardinal;
  TotalUncommitted
  : Cardinal;
  TotalCommitted : Cardinal;
  TotalAllocated : Cardinal
  ;
  TotalFree : Cardinal;
  FreeSmall : Cardinal;
  FreeBig : Cardinal
  ;
  Unused : Cardinal;
  Overhead : Cardinal;
  HeapErrorCode : Cardinal
  ;
end
```

THeapStatus is the record describing the current heap status. It is returned by the GetHeapStatus (1474) call.

```
TInitThreadVarHandler = procedure(var offset: DWord; size: DWord)
```

Threadvar initialization callback type for TThreadManager ([1400](#)).

TInterfacedClass = Class of TInterfacedObject

TInterfacedClass is a descendent of

```
tinterfaceentrytype = (etStandard, etVirtualMethodResult,
  etStaticMethodResult, etFieldValue,
  etVirtualMethodClass, etStaticMethodClass,
  etFieldValueClass)
```

Table 75.15: Enumeration values for type tinterfaceentrytype

Value	Explanation
etFieldValue	Field value.
etFieldValueClass	Interface provided by a class field.
etStandard	Standard entry.
etStaticMethodClass	Interface provided by a static class method.
etStaticMethodResult	Static method.
etVirtualMethodClass	Interface provided by a virtual class method.
etVirtualMethodResult	Virtual method.

This is an internal type for the compiler to encode calls to dispatch interfaces.

```
tinterfacetable = record
public
  EntryCount : SizeUInt;
  Entries
    : Array[0..0] of tinterfaceentry;
end
```

Record to store list of interfaces of a class.

TLibHandle = PtrInt

TLibHandle should be considered an opaque type. It is defined differently on various platforms. The definition shown here depends on the platform for which the documentation was generated.

TLineEndStr = string

TLineEndStr is an alias for the actual line ending string type, used in TextRec ([1389](#)). It should be treated as opaque.

```
TLoadLibraryAHandler = function(const Name: RawByteString) :
  TLibHandle
```

TLoadLibraryAHandler is the type for the loadlibrary call using ansistring names in the TDynLibsManager ([1387](#)) dynamic library loading manager.

```
TLoadLibraryUHandler = function(const Name: UnicodeString) :
  TLibHandle
```

TLoadLibraryUHandler is the type for the loadlibrary call using unicode names in the TDynLibsManager (1387) dynamic library loading manager.

```
TMemoryManager = record
public
  NeedLock : Boolean;
  Getmem : function
    (Size: PtrUInt) : Pointer;
  Freemem : function(p: pointer) : PtrUInt
  ;
  FreememSize : function(p: pointer; Size: PtrUInt) : PtrUInt;
  AllocMem : function(Size: PtrUInt) : Pointer;
  ReAllocMem : function
    (var p: pointer; Size: PtrUInt) : Pointer;
  MemSize : function(p
    : pointer) : PtrUInt;
  InitThread : procedure;
  DoneThread : procedure
  ;
  RelocateHeap : procedure;
  GetHeapStatus : function : THeapStatus
  ;
  GetFPCHeapStatus : function : TFPCHeapStatus;
end
```

TMemoryManager describes the memory manager. For more information about the memory manager, see the programmer's reference.

```
TMethod = record
public
  Code : CodePointer;
  Data : Pointer;
end
```

TMethod describes a general method pointer, and is used in Run-Time Type Information handling.

```
TMsgStrTable = record
public
  name : PShortString;
  method : CodePointer
  ;
end
```

Record used in string message handler table.

```
TOpaqueData = record
end
```

TOpaqueData represents data for which the internal structure must not be known. It is mostly useful for the pointer definition POpaqueData (1377).

TOrdinalEntry = SizeUInt

Ordinal of entry point (windows only).

TOSTimestamp = Int64

packed TPCCharArray = Array[0..(MaxLongintdivSizeOf(PChar))-1] of
PChar

Array of PChar.

TProcedure = procedure

Simple procedural type.

TReleaseThreadVarsHandler = procedure

Threadvar release callback type for TThreadManager (1400).

TRelocateThreadVarHandler = function(offset: DWord) : pointer

Threadvar relocation callback type for TThreadManager (1400).

TResourceHandle = PtrUInt

This is an opaque type.

```
TResourceManager = record
public
  HINSTANCEFunc : function : TFPResourceHMODULE
  ;
  EnumResourceTypesFunc : function(ModuleHandle: TFPResourceHMODULE
  ; EnumFunc: EnumResTypeProc;
    lParam: PtrInt) : LongBool
  ;
  EnumResourceNamesFunc : function(ModuleHandle: TFPResourceHMODULE
  ; ResourceType: PChar;
    EnumFunc: EnumResNameProc; lParam
  : PtrInt) : LongBool;
  EnumResourceLanguagesFunc : function(ModuleHandle
  : TFPResourceHMODULE; ResourceType: PChar;
    ResourceName
  : PChar; EnumFunc: EnumResLangProc; lParam: PtrInt)
  :
  LongBool;
  FindResourceFunc : function(ModuleHandle: TFPResourceHMODULE
  ; ResourceName: PChar;
    ResourceType: PChar) : TFPResourceHandle
  ;
  FindResourceExFunc : function(ModuleHandle: TFPResourceHMODULE
  ; ResourceType: PChar;
```

```

        ResourceName: PChar; Language:
Word) : TFPResourceHandle;
LoadResourceFunc : function(ModuleHandle
: TFPResourceHMODULE; ResHandle: TFPResourceHandle)
:
TFPResourceHGLOBAL;
SizeofResourceFunc : function(ModuleHandle
: TFPResourceHMODULE; ResHandle: TFPResourceHandle)
:
LongWord;
LockResourceFunc : function(ResData: TFPResourceHGLOBAL
) : Pointer;
UnlockResourceFunc : function(ResData: TFPResourceHGLOBAL
) : LongBool;
FreeResourceFunc : function(ResData: TFPResourceHGLOBAL
) : LongBool;
end

```

TResourceManager is the record describing the resource manager. Depending on the kind of resources (internal, external), another resource managing handler is installed by the system. The resource manager record is used by all resource handling functions to do the actual work: for each function in the API, a handler function is available. People wishing to implement their own resource manager, must implement all handler functions in their implementation.

As soon as resources are used, the compiler will install a resource manager, depending on the platform, this may be an internal or an external resource manager.

```
TRTLCreateEventHandler = function : PRTLEvent
```

Callback type for creating a TRTLEvent type in TThreadManager (1400).

```
TRTLCRITICALSECTION = Opaque type
```

TRTLCriticalSection represents a critical section (a mutex). This is an opaque type, it can differ from operating system to operating system. No assumptions should be made about its structure or contents.

```
TRTLEventHandler = procedure(AEvent: PRTLEvent)
```

Generic TRTLEvent handling type for TThreadManager (1400).

```
TRTLEventHandlerTimeout = procedure(AEvent: PRTLEvent; timeout: LongInt
)
```

TRTLEvent timeout handling type for TThreadManager (1400).

```
trtlmethod = procedure of object
```

Callback type for synchronization event.

```
TRuntimeError = (reNone, reOutOfMemory, reInvalidPtr, reDivByZero,
reRangeError, reIntOverflow, reInvalidOp, reZeroDivide
,

```

```

reOverflow, reUnderflow, reInvalidCast, reAccessViolation
,
rePrivInstruction, reControlBreak, reStackOverflow
,
reVarTypeCast, reVarInvalidOp, reVarDispatch,
reVarArrayCreate, reVarNotArray, reVarArrayBounds,
reAssertionFailed, reExternalException, reIntfCastError
,
reSafeCallError, reQuit, reCodesetConversion,
reNoDynLibsSupport, reThreadError)

```

Table 75.16: Enumeration values for type TRuntimeError

Value	Explanation
reAccessViolation	Access Violation.
reAssertionFailed	Assertion failed error.
reCodesetConversion	Code set conversion error.
reControlBreak	User pressed CTRL-C.
reDivByZero	Division by zero error.
reExternalException	An external exception occurred.
reIntfCastError	Interface typecast error.
reIntOverflow	Integer overflow error.
reInvalidCast	Invalid (class) typecast error.
reInvalidOp	Invalid operation error.
reInvalidPtr	Invalid pointer error.
reNoDynLibsSupport	Runtime error if no dynamic library support is available.
reNone	No error.
reOutOfMemory	Out of memory error.
reOverflow	Overflow error.
rePrivInstruction	Privileged instruction error.
reQuit	Quit signal error.
reRangeError	Range check error.
reSafeCallError	Safecall (IDispInterface) error.
reStackOverflow	Stack overflow error.
reThreadError	Runtime error if no thread support is available.
reUnderflow	Underflow error.
reVarArrayBounds	Variant array bounds error.
reVarArrayCreate	Variant array creation error.
reVarDispatch	Variant Dispatch error.
reVarInvalidOp	Invalid variant operation error.
reVarNotArray	Variant is not an array error.
reVarTypeCast	Invalid typecase from variant.
reZeroDivide	Division by zero error.

TRuntimeError is used in the Error ([1459](#)) procedure to indicate what kind of error should be reported.

TSafeCallErrorProc = procedure(error: HRESULT; addr: pointer)

Prototype of a safecall error handler routine. Error is the error number (passed by the Windows operating system) and Addr is the address where the error occurred.

```
TSemaphoreDestroyHandler = procedure(const sem: Pointer)
```

TSemaphoreDestroyHandler is the function prototype to destroy an existing semaphore, as returned by (ThreadManager.SemaphoreInit). It is used by the thread manager (ThreadManager.SemaphoreDest

```
TSemaphorePostHandler = procedure(const sem: Pointer)
```

TSemaphorePostHandler is the function prototype to post an event to the semaphore. It should handle a pointer as returned by the ThreadManager.SemaphoreInit procedure. it's used by the thread manager ThreadManager.SemaphorePost.

```
TSemaphoreWaitHandler = procedure(const sem: Pointer)
```

TSemaphoreWaitHandler is the function prototype to wait on an event on the semaphore (which should be posted to the semaphore with ThreadManager.SemaphorePost). It should handle a pointer as returned by the ThreadManager.SemaphoreInit procedure. it's used by the thread manager ThreadManager.SemaphoreWait.

```
TSemaphoreInitHandler = function : Pointer
```

TSemaphoreInitHandler is the function prototype for initializing a semaphore. It is used by the thread manager (ThreadManager.SemaphoreInit) to create semaphores. The function should return a pointer, usable by the other semaphore functions of the thread manager.

```
TStandardCodePageEnum = (scpAnsi, scpConsoleInput, scpConsoleOutput
    ,
    scpFileSystemSingleByte)
```

Table 75.17: Enumeration values for type TStandardCodePageEnum

Value	Explanation
scpAnsi	Ansi codepage (CP_ACP).
scpConsoleInput	Console input codepage.
scpConsoleOutput	Console output codepage.
scpFileSystemSingleByte	File system single byte codepage.

TStandardCodePageEnum describes several types of standard used codepages, which can be queried by the unicode string manager TUnicodeStringManager ([1403](#)).

```
TStringMessageTable = record
public
    count : LongInt;
    msgstrtable
        : Array[0..0] of TMsgStrTable;
end
```

Record used to describe the string messages handled by a class. It consists of a count, followed by an array of TMsgStrTable ([1394](#)) records.

```
TSystemCodePage = Word
```

TSystemCodePage is a type used to indicate code pages. It should be treated as an opaque type.

TTextBuf = TextBuf

TTextBuf is an alias for TextBuf

TTextLineBreakStyle = (tlbsLF,tlbsCRLF,tlbsCR)

Table 75.18: Enumeration values for type TTextLineBreakStyle

Value	Explanation
tlbsCR	Carriage-return (#13, Mac-OS style).
tlbsCRLF	Carriage-return, line-feed (#13#30, Windows style).
tlbsLF	Line-feed only (#10, Unix style).

Text line break style. (end of line character).

TThreadFunc = function(parameter: pointer) : PtrInt

Thread function prototype.

TThreadGetPriorityHandler = function(threadHandle: TThreadID) :
LongInt

Callback type for thread priority getting in TThreadManager (1400).

TThreadHandler = function(threadHandle: TThreadID) : DWord

Generic thread handler callback for TThreadManager (1400).

TThreadID = PtrUInt

This is an opaque type, it can differ from operating system to operating system.

```
TThreadManager = record
public
    InitManager : function : Boolean
    ;
    DoneManager : function : Boolean;
    BeginThread : TBeginThreadHandler
    ;
    EndThread : TEndThreadHandler;
    SuspendThread : TThreadHandler
    ;
    ResumeThread : TThreadHandler;
    KillThread : TThreadHandler;
    CloseThread : TThreadHandler;
    ThreadSwitch : TThreadSwitchHandler
    ;
    WaitForThreadTerminate : TWaitForThreadTerminateHandler;
```



```

ThreadSetPriority
: TThreadSetPriorityHandler;
ThreadGetPriority : TThreadGetPriorityHandler
;
GetCurrentThreadId : TGetCurrentThreadIdHandler;
SetThreadDebugNameA
: TThreadSetThreadDebugNameHandlerA;
SetThreadDebugNameU : TThreadSetThreadDebugNameHandlerU
;
InitCriticalSection : TCriticalSectionHandler;
DoneCriticalSection
: TCriticalSectionHandler;
EnterCriticalSection : TCriticalSectionHandler
;
TryEnterCriticalSection : TCriticalSectionHandlerTryEnter;
LeaveCriticalSection
: TCriticalSectionHandler;
InitThreadVar : TInitThreadVarHandler
;
RelocateThreadVar : TRelocateThreadVarHandler;
AllocateThreadVars
: TAllocateThreadVarsHandler;
ReleaseThreadVars : TReleaseThreadVarsHandler
;
BasicEventCreate : TBasicEventCreateHandler;
BasicEventDestroy
: TBasicEventHandler;
BasicEventResetEvent : TBasicEventHandler
;
BasicEventSetEvent : TBasicEventHandler;
BasicEventWaitFor
: TBasicEventWaitForHandler;
RTLEventCreate : TRTLCreateEventHandler
;
RTLEventDestroy : TRTLEventHandler;
RTLEventSetEvent : TRTLEventHandler
;
RTLEventResetEvent : TRTLEventHandler;
RTLEventWaitFor : TRTLEventHandler
;
RTLEventWaitForTimeout : TRTLEventHandlerTimeout;
end

```

`TThreadManager` is a record that contains all callbacks needed for the thread handling routines of the Free Pascal Run-Time Library. The thread manager can be set by the `SetThreadManager` (1544) procedure, and the current thread manager can be retrieved with the `GetThreadManager` (1477) procedure.

The Windows RTL will set the thread manager automatically to a system thread manager, based on the Windows threading routines. Unix operating systems provide a unit `cthreads` which implements threads based on the C library POSIX thread routines. It is not included by default, because it would make the system unit dependent on the C library.

For more information about thread programming, see the programmer's guide.

```
TThreadSetPriorityHandler = function(threadHandle: TThreadID;
  Prio: LongInt) : Boolean
```

Callback type for thread priority setting in TThreadManager (1400).

```
TThreadSetThreadDebugNameHandlerA = procedure(threadHandle: TThreadID
  ;
                                     const ThreadName
  : AnsiString)
```

TThreadSetThreadDebugNameHandlerA is the type of the TThreadManager (1400)SetThreadDebugNameA method. It receives the thread ID in threadHandle and the name (as AnsiString) to apply in ThreadName.

```
TThreadSetThreadDebugNameHandlerU = procedure(threadHandle: TThreadID
  ;
                                     const ThreadName
  : UnicodeString)
```

TThreadSetThreadDebugNameHandlerU is the type of the TThreadManager (1400)SetThreadDebugNameU method. It receives the thread ID in threadHandle and the name to apply (as unicodeString) in ThreadName.

```
TThreadSwitchHandler = procedure
```

Callback type for thread switch in TThreadManager (1400).

```
TTime = TDateTime
```

TTime is defined for Delphi compatibility. This type is deprecated, use TDateTime (1386) instead.

```
TTypeKind = (tkUnknown,tkInteger,tkChar,tkEnumeration,tkFloat,tkSet
  ,
               tkMethod,tkSString,tkLString,tkAString,tkWString,tkVariant
  ,
               tkArray,tkRecord,tkInterface,tkClass,tkObject,tkWChar
  ,
               tkBool,tkInt64,tkQWord,tkDynArray,tkInterfaceRaw,tkProcVar
  ,
               tkUString,tkUChar,tkHelper,tkFile,tkClassRef,tkPointer
  )
```

Table 75.19: Enumeration values for type TTypeKind

Value	Explanation
tkArray	Array property.
tkAString	Ansistring property.
tkBool	Boolean property.
tkChar	Char property.
tkClass	Class property.
tkClassRef	Class reference type.
tkDynArray	Dynamic array property.
tkEnumeration	Enumeration type property.
tkFile	File type.
tkFloat	Float property.
tkHelper	Helper type.
tkInt64	Int64 property.
tkInteger	Integer property.
tkInterface	Interface property.
tkInterfaceRaw	Raw interface property.
tkLString	Longstring property.
tkMethod	Method property.
tkObject	Object property.
tkPointer	Generic pointer type.
tkProcVar	Procedural variable.
tkQWord	QWord property.
tkRecord	Record property.
tkSet	Set property.
tkSString	Shortstring property.
tkUChar	Unicode character.
tkUnknown	Unknown property type.
tkUString	Unicode string.
tkVariant	Variant property.
tkWChar	Widechar property.
tkWString	Widestring property.

Type of a property or value.

```
TUCS4CharArray = Array[0..$efffffff] of UCS4Char
```

Array of UCS4Char (1412) characters.

```
TUnicodeStringManager = record
public
  Wide2AnsiMoveProc : procedure
    (source: PWideChar; var dest: RawByteString;
     cp: TSystemCodePage
    ; len: SizeInt);
  Ansi2WideMoveProc : procedure(source: PChar; cp
    : TSystemCodePage; var dest: widestring;
     len: SizeInt
  );
  UpperWideStringProc : function(const S: WideString) : WideString
  ;
```

```

LowerWideStringProc : function(const S: WideString) : WideString
;
CompareWideStringProc : function(const s1: WideString; const s2
: WideString;
    Options: TCompareOptions) : PtrInt;
CharLengthPCharProc
: function(const Str: PChar) : PtrInt;
CodePointLengthProc : function
(const Str: PChar; MaxLookAead: PtrInt) : PtrInt;
UpperAnsiStringProc
: function(const s: ansistring) : ansistring;
LowerAnsiStringProc
: function(const s: ansistring) : ansistring;
CompareStrAnsiStringProc
: function(const S1: ansistring; const S2: ansistring) : PtrInt;
CompareTextAnsiStringProc : function(const S1: ansistring; const
S2: ansistring) : PtrInt;
StrCompAnsiStringProc : function(S1:
PChar; S2: PChar) : PtrInt;
StrICompAnsiStringProc : function(S1
: PChar; S2: PChar) : PtrInt;
StrLCompAnsiStringProc : function
(S1: PChar; S2: PChar; MaxLen: PtrUInt) : PtrInt;
StrLICompAnsiStringProc
: function(S1: PChar; S2: PChar; MaxLen: PtrUInt) : PtrInt;
StrLowerAnsiStringProc
: function(Str: PChar) : PChar;
StrUpperAnsiStringProc : function
(Str: PChar) : PChar;
ThreadInitProc : procedure;
ThreadFiniProc
: procedure;
Unicode2AnsiMoveProc : procedure(source: PUnicodeChar
; var dest: RawByteString;
    cp: TSystemCodePage; len:
SizeInt);
Ansi2UnicodeMoveProc : procedure(source: PChar; cp: TSystemCodePage
; var dest: unicodestring;
    len: SizeInt);
UpperUnicodeStringProc
: function(const S: UnicodeString) : UnicodeString;
LowerUnicodeStringProc
: function(const S: UnicodeString) : UnicodeString;
CompareUnicodeStringProc
: function(const s1: UnicodeString; const s2: UnicodeString;
Options: TCompareOptions) : PtrInt;
GetStandardCodePageProc
: function(const stdcp: TStandardCodePageEnum) : TSystemCodePage
;
end

```

TUnicodeStringManager is currently the same as the TWideStringManager ([1412](#)) manager record. It performs the same functions: converting Unicode strings to ansistrings and vice-versa, performing uppercase to lowercase transformations and comparing strings.

```
TUnloadLibraryHandler = function(Lib: TLibHandle) : Boolean
```

TUnloadLibraryHandler is the type for the UnloadLibrary call using anistring names in the TDynLibsManager (1387) dynamic library loading manager.

```
tvararray = record
public
  dimcount : Word;
  flags : Word;
  elementsize
  : LongInt;
  lockcount : LongInt;
  data : pointer;
  bounds : tvararrayboundarray
;
end
```

tvararray is a record describing a variant array. It contains some general data, followed by a number of TVarArrayBound (1404) records equal to the number of dimensions in the array (dimcount).

```
tvararraybound = record
public
  elementcount : LongInt;
  lowbound
  : LongInt;
end
```

tvararraybound is used to describe one dimension in a variant array.

```
tvararrayboundarray = Array[0..0] of tvararraybound
```

array of tvararraybound (1404) records.

```
tvararraycoorarray = Array[0..0] of LongInt
```

Array of variant array coordinates.

```
tvardata = packed record
public
  vtype : tvartype;
case Integer of
  0: (
public
  res1 : Word;
case Integer of
  0: (
public
  res2 : Word
;
  res3 : Word;
case Word of
varsmallint: (
```

```
public
    vsmallint :
        SmallInt;
);
varinteger: (
public
    vinteger : LongInt;
);
varsingle
    : (
public
    vsingle : single;
);
vardouble: (
public
    vdouble :
        Double;
);
vardate: (
public
    vdate : TDateTime;
);
varcurrency
    : (
public
    vcurrency : currency;
);
varolestr: (
public
    volestr
        : PWideChar;
);
vardispatch: (
public
    vdispatch : pointer;
);
varerror
    : (
public
    verror : HRESULT;
);
varboolean: (
public
    vboolean
        : wordbool;
);
varunknown: (
public
    vunknown : pointer;
);
varustring
    : (
public
    vustring : pointer;
```

```
);
varshortint: (
public
    vshortint
    : ShortInt;
);
varbyte: (
public
    vbyte : Byte;
);
varword: (
public
    vword : Word;
);
varlongword: (
public
    vlongword : DWord;
);
varint64: (
public
    vint64 : Int64;
);
varqword: (
public
    vqword
    : QWord;
);
varword64: (
public
    vword64 : QWord;
);
varstring:
(
public
    vstring : pointer;
);
varany: (
public
    vany : pointer
    ;
);
vararray: (
public
    varray : pvararray;
);
varbyref: (
public
    vpointer : pointer;
);
varrecord: (
public
    vrecord : pointer
    ;
    precinfo : pointer;
```

```

);
);
1: (
public
    vlongs : Array[0..2] of
        LongInt;
);
);
1: (
public
    vwords : Array[0..6] of Word;
);
2:
(
public
    vbytes : Array[0..13] of Byte;
);
end

```

TVarData is a record representation of a variant. It contains the internal structure of a variant and is handled by the various variant handling routines.

```

tvariantmanager = record
public
    vartoint : function(const v: variant
    ) : LongInt;
    vartoint64 : function(const v: variant) : Int64;
    vartoword64 : function(const v: variant) : QWord;
    vartobool : function
    (const v: variant) : Boolean;
    vartoreal : function(const v: variant
    ) : extended;
    vartotdatetime : function(const v: variant) : TDateTime
    ;
    vartocurr : function(const v: variant) : currency;
    vartopstr
    : procedure(var s; const v: variant);
    vartolstr : procedure(var
    s: ansistring; const v: variant);
    vartowstr : procedure(var s:
    wideststring; const v: variant);
    vartointf : procedure(var intf:
    IInterface; const v: variant);
    vartodisp : procedure(var disp:
    IDispatch; const v: variant);
    vartodynarray : procedure(var dynarr
    : pointer; const v: variant; typeinfo: pointer);
    varfrombool : procedure
    (var dest: variant; const source: Boolean);
    varfromint : procedure
    (var dest: variant; const source: LongInt;
        const Range
    : LongInt);
    varfromint64 : procedure(var dest: variant; const source

```



```

: Int64);
varfromword64 : procedure(var dest: variant; const source
: QWord);
varfromreal : procedure(var dest: variant; const source
: extended);
varfromdatetime : procedure(var dest: Variant; const
source: TDateTime);
varfromcurr : procedure(var dest: Variant;
const source: Currency);
varfrompstr : procedure(var dest: variant
; const source: ShortString);
varfromlstr : procedure(var dest:
variant; const source: ansistring);
varfromwstr : procedure(var
dest: variant; const source: WideString);
varfromintf : procedure
(var dest: variant; const source: IInterface);
varfromdisp : procedure
(var dest: variant; const source: IDispatch);
varfromdynarray :
procedure(var dest: variant; const source: pointer; typeinfo: pointer
);
olevarfrompstr : procedure(var dest: olevariant; const source
: shortstring);
olevarfromlstr : procedure(var dest: olevariant
; const source: ansistring);
olevarfromvar : procedure(var dest
: olevariant; const source: variant);
olevarfromint : procedure
(var dest: olevariant; const source: Int64;
const range
: ShortInt);
varop : procedure(var left: variant; const right: variant
; opcode: tvarop);
cmpop : function(const left: variant; const right
: variant;
const opcode: tvarop) : Boolean;
varneg
: procedure(var v: variant);
varnot : procedure(var v: variant)
;
varinit : procedure(var v: variant);
varclear : procedure(var
v: variant);
varaddref : procedure(var v: variant);
varcopy
: procedure(var dest: variant; const source: variant);
varcast
: procedure(var dest: variant; const source: variant; vartype: LongInt
);
varcastole : procedure(var dest: variant; const source: variant
; vartype: LongInt);
dispinvoke : procedure(dest: pvardata; var
source: tvardata; calldesc: pcalldesc);

```

```

        params: pointer
    );
    vararrayredim : procedure(var a: variant; highbound: SizeInt
    );
    vararrayget : function(const a: variant; indexcount: SizeInt
    ; indices: PLongint)
        : variant;
    vararrayput : procedure
    (var a: variant; const value: variant; indexcount: SizeInt;
    indices: PLongint);
    writevariant : function(var t: text
    ; const v: variant; width: LongInt) : Pointer;
    write0Variant : function
    (var t: text; const v: Variant) : Pointer;
end

```

TVariantManager describes the variant manager as expected by the SetVariantManager (1544) call.

```

tvarop = (opadd,opsubtract,opmultiply,opdivide,opintdivide,opmodulus
,
        opshiftleft,opshiftright,opand,opor,opxor,opcompare,opnegate
,
        opnot,opcmpeq,opcmpne,opcmplt,opcmple,opcmpgt,opcmpge
,oppower)

```

Table 75.20: Enumeration values for type tvarop

Value	Explanation
opadd	Variant operation: Addition.
opand	Variant operation: Binary AND operation.
opcmpeq	Variant operation: Compare equal.
opcmpge	Variant operation: Compare larger than or equal.
opcmpgt	Variant operation: Compare larger than.
opcmple	Variant operation: Compare less than or equal to.
opcmplt	Variant operation: Compare less than.
opcmpne	Variant operation: Compare not equal.
opcompare	Variant operation: Compare.
opdivide	Variant operation: division.
opintdivide	Variant operation: integer divide.
opmodulus	Variant operation: Modulus.
opmultiply	Variant operation: multiplication.
opnegate	Variant operation: negation.
opnot	Variant operation: Binary NOT operation.
opor	Variant operation: Binary OR operation.
oppower	Variant operation: Power.
opshiftleft	Variant operation: Shift left.
opshiftright	Variant operation: Shift right.
opsubtract	Variant operation: Subtraction.
opxor	Variant operation: binary XOR operation.

tvarop describes a variant operation. It is mainly used for the variant manager to implement the various conversions and mathematical operations on a variant.

```

TVarRec = record
case VType : SizeInt of
vtInteger: (
public
  VInteger
    : LongInt;
);
vtBoolean: (
public
  VBoolean : Boolean;
);
vtChar
  : (
public
  VChar : Char;
);
vtWideChar: (
public
  VWideChar : WideChar
  ;
);
vtExtended: (
public
  VExtended : PExtended;
);
vtString: (
  public
  VString : PShortString;
);
vtPointer: (
public
  VPointer
    : Pointer;
);
vtPChar: (
public
  VPChar : PAnsiChar;
);
vtObject
  : (
public
  VObject : TObject;
);
vtClass: (
public
  VClass : TClass
  ;
);
vtPWideChar: (
public
  VPWideChar : PWideChar;

```

```

);
vtAnsiString
  : (
public
  VAnsiString : Pointer;
);
vtCurrency: (
public
  VCurrency
  : PCurrency;
);
vtVariant: (
public
  VVariant : PVariant;
);
vtInterface
  : (
public
  VInterface : Pointer;
);
vtWideString: (
public
  VWideString
  : Pointer;
);
vtInt64: (
public
  VInt64 : PInt64;
);
vtUnicodeString
  : (
public
  VUnicodeString : Pointer;
);
vtQWord: (
public
  VQWord
  : PQWord;
);
end

```

TVarRec is a record generated by the compiler for each element in a array of const call. The procedure that receives the constant array receives an array of TVarRec elements, with lower bound zero and high bound equal to the number of elements in the array minus one (as returned by High(Args))

tvartype = Word

Type with size of variant type.

```

TWaitForThreadTerminateHandler = function(threadHandle: TThreadID
;
                                     TimeoutMs: LongInt)
: DWord

```

Callback type for thread termination in TThreadManager ([1400](#)).

TWideStringManager = TUnicodeStringManager

TWideStringManager contains the definition of the widestring manager.

UCS2Char = WideChar

UCS2 Unicode character.

UCS4Char =

UCS Unicode character (unsigned 32 bit word).

UCS4String = Array of UCS4Char

String of UCS4Char ([1412](#)) characters.

UInt16 = Word

An unsigned 16-bits integer.

UInt32 = Cardinal

An unsigned 32-bits integer.

UInt64 = QWord

Unsigned 64-bit integer.

UInt8 = Byte

An unsigned 8-bits integer.

UIntPtr = PtrUInt

Alias for PtrUInt ([1381](#)) type for compatibility with later Delphi versions.

UnicodeChar = WideChar

UnicodeChar is a single character from a UnicodeString. It equals WideChar in all respects.

UnicodeString = UnicodeString

UnicodeString is a string of WideChars. The main difference with WideString is that unicode-string is reference counted, and WideString is not reference counted on Windows.

UTF8Char = AnsiChar

UTF8Char is provided for completeness, a type alias for AnsiChar.

`UTF8String = ansistring`

UTF-8 Unicode (Ansi) string.

`ValReal = Extended`

`ValReal` is an alias for the largest available floating point type on the architecture the program runs on. On most processors, it should be one of `Double` or `Extended`.

`ValSInt = LongInt`

Integer with the same size as the return code of the `Val` (1570) function.

`ValSInt = Int64`

Integer with the same size as the return code of the `Val` (1570) function.

`ValUInt = QWord`

Integer with the same size as the return code of the `Val` (1570) function.

`ValUInt = Cardinal`

Integer with the same size as the return code of the `Val` (1570) function.

`WChar = WideChar`

Wide char (16-bit sized char).

`WideChar = #$0000..#$FFFF`

This type is the base unit for all two byte character types, like `UnicodeString` (1412) and `WideString` (1413)

`WideString = WideString`

`WideString` is an alias for `UnicodeString` on UNIX. On windows, it is a different type which has no reference counting.

`Word = 0..65535`

An unsigned 16-bits integer.

75.11.3 Variables

`argc : LongInt; external 'operatingsystem_parameter_argc'`

`argc` contains the number of command-line arguments passed to the program by the OS. It is not available on all systems.

`argv : PPChar; external 'operatingsystem_parameter_argv'`

`argv` contains a pointer to a `nil`-terminated array of null-terminated strings, containing the command-line arguments passed to the program by the OS. It is not available on all systems.

`DefaultFileSystemCodePage` : `TSystemCodePage`

`DefaultFileSystemCodePage` determines the code page to which file/path names are translated before they are passed to OS API calls, if the RTL uses a single byte OS API for this purpose on the current platform.

This code page is also used for intermediate operations on file paths inside the RTL before making OS API calls.

This variable does not exist in Delphi, and has been introduced in FPC to make it possible to change the value of `DefaultSystemCodePage` without breaking RTL interfaces with the OS file system API calls.

The initial value of this variable depends on the platform:

- Windows: UTF-8, because the RTL uses UTF-16 OS API calls (so no data is lost in intermediate operations).
- OS X and iOS: UTF-8 (as defined by Apple)
- Unix (excluding OS X and iOS): equals `DefaultSystemCodePage` (1415). This is because the encoding of file names is undefined on Unix platforms: it is an untyped array of bytes that can be interpreted in any way; Specifically, it is not guaranteed to be valid UTF-8.
- Other platforms: same as `DefaultSystemCodePage` (1415).

The value of this variable may be changed using the `SetMultiByteFileSystemCodePage` (1540) procedure.

Remark The Unix/OS X/iOS settings only apply in case the `cwstring` widestring manager is installed, otherwise `DefaultFileSystemCodePage` will have the same value as `DefaultSystemCodePage` after program startup.

`DefaultRTLFileSystemCodePage` : `TSystemCodePage`

`DefaultRTLFileSystemCodePage` determines the code page to which file/path names are translated before they are returned from `RawByteString` (1383) file/path RTL routines.

Examples include the file/path names returned by the `RawByteString` versions of `SysUtils.FindFirst` (1335) and `GetDir` (1473).

The main reason for its existence is to enable the RTL to provide backward compatibility with earlier versions of FPC, as these always returned strings encoded in whatever the OS' single byte API used (normally `DefaultSystemCodePage` (1415)).

The initial value of this variable depends on the platform:

- Windows: `DefaultSystemCodePage`, for backward compatibility.
- OS X and iOS: UTF-8, for backward compatibility. It was already always UTF-8 in the past, since that's what the OS file APIs returned, and the data was never converted.
- Other Unixes: `DefaultSystemCodePage`, for the same reason as `DefaultFileSystemCodePage` (1414). Setting this to a different value than `DefaultFileSystemCodePage` is a bad idea on these platforms, since any code page conversion can corrupt these strings as their initial encoding is unknown.

- Other platforms: same as `DefaultSystemCodePage`.

The value of this variable can be set using the `SetMultiByteRTLFileSystemCodePage` (1540) call.

`DefaultSystemCodePage` : `TSystemCodePage`

`DefaultSystemCodePage` is used to determine how `CP_ACP` is interpreted; it is what the program considers to be the current system codepage.

It is initialized to the default system codepage.

- On windows, this is the result of the `GetACP` operating call, which returns the Windows ANSI code page.
- On iOS, this is UTF-8
- on other Unixes this will be based on the currently set `LANG` or `LC_CTYPE` environment variables. Normally this is UTF-8, but that is not guaranteed to be the case.
- For all other platforms it is set to `CP_ACP`, as these platforms currently do not support multiple code pages, and are hardcoded to use their OS-specific code page in all cases.

The `DefaultSystemCodePage` value may be set using `SetMultiByteConversionCodePage` (1540). That means that it is not a good idea to use its value to determine the real OS "default system code page".

Note that if you change `DefaultSystemCodePage`, you should call `TEncoding.FreeEncodings` to free the encodings that were created. They will be recreated with the correct (new) `DefaultSystemCodePage`.

`DefaultUnicodeCodePage` : `TSystemCodePage`

`DefaultUnicodeCodePage` is the unicode code page for a new unicode string. On most platforms, this is `CP_UTF16` (1343).

`DispCallByIDProc` : `CodePointer`

`VarDispProc` is called by the compiler if it needs to perform an interface call from a variant which contains a dispatch interface. For instance, the following call:

```
Var
  V : OleVariant;
begin
  (V as IWord).OpenDocument('c:\temp\mydoc.doc');
end;
```

where `IWord` is a dispatch interface is encoded by the compiler and passed to `DispCallByIDProc`.

This pointer must be set by a routine that calls the OS COM handling routines.

`envp` : `PPChar`; external 'operatingsystem_parameter_envp'

`envp` contains a pointer to a nil-terminated array of null-terminated strings, containing the environment variables passed to the program by the OS. It is not available on all systems.

`ErrOutput` : `Text`

ErrOutput is provided for Delphi compatibility.

```
ExitCode : LongInt; public 'operatingsystem_result'
```

Exit code for the program, will be communicated to the OS on exit.

```
FirstDotAtFileNameStartIsExtension : Boolean = False
```

FirstDotAtFileNameStartIsExtension determines what happens if a filename starts with a dot (.) character. If True, then the whole file name will be treated as extension. If False, then the extension is empty.

```
InOutRes : Word
```

InOutRes contains the result of the last I/O operation using one of the file I/O routines. When I/O checks are enabled, this variable is checked and if it is nonzero, a runtime error is raised.

Although it is a variable for historical reasons, it is not meant to be written to by user code, outside of routines that implement some form of file I/O.

```
Input : Text
```

Standard input text file.

```
IsConsole : Boolean; public 'operatingsystem_isconsole' = False
```

True for console applications, False for GUI applications.

```
IsLibrary : Boolean; public 'operatingsystem_islibrary' = False
```

True if the current module is a library. Otherwise module is an executable.

```
mem : Array[0..$7fffffff-1] of Byte
```

mem is an array of bytes, representing the computer's memory. This array is available only when compiling for the Dos Go32V2 target. It's use is not recommended, and it is not even available on other platforms.

```
meml : Array[0..($7fffffffdivsizeof(longint))-1] of LongInt
```

memw is an array of longints, representing the computer's memory as 32-bit signed integers. This array is available only when compiling for the Dos Go32V2 target. It's use is not recommended, and it is not even available on other platforms.

```
memw : Array[0..($7fffffffdivsizeof(word))-1] of Word
```

memw is an array of words, representing the computer's memory as 2-byte words. This array is available only when compiling for the Dos Go32V2 target. It's use is not recommended, and it is not even available on other platforms.

```
NoErrMsg : Boolean = Falseplatform
```

Unused, for Delphi compatibility.

Output : Text

Standard output text file.

RandSeed : Cardinal

Seed for Random ([1518](#)) function.

ReturnNilIfGrowHeapFails : Boolean

ReturnNilIfGrowHeapFails describes what happens if there is no more memory available from the operating system. if set to True the memory manager will return Nil. If set to False then a run-time error will occur.

softfloat_exception_flags : TFPUEExceptionMask

Current soft float exception flags.

softfloat_exception_mask : TFPUEExceptionMask

Current soft float exception mask.

softfloat_rounding_mode : TFPURoundingMode

softfloat_rounding_mode determines how the software floating-point emulation routines do the rounding. The value can be one of the following:

float_round_nearest_even Round to nearest even number.

float_round_down Round down.

float_round_up Round up.

float_round_to_zero Round in the direction of zero (down for positive, up for negative).

StackBottom : Pointer

Current stack bottom.

StackLength : SizeUInt

Maximum stack length.

StdErr : Text

Standard diagnostic output text file.

StdOut : Text

Alias for Output ([1417](#)).

ThreadID : TThreadID

Current Thread ID.

UTF8CompareLocale : TSystemCodePage

UTF8CompareLocale is currently present for Delphi compatibility only, it is not used in FPC code.

widestringmanager : TUnicodeStringManager

Contains the current widestring manager. Do not use directly.

WriteErrorsToStdErr : Boolean = True

WriteErrorsToStdErr can be set to True to write error messages (run-time errors, exceptions) to StdErr instead of to standard output. This is the default behaviour. When set to False, error message will be written to standard output.

75.12 Procedures and functions

75.12.1 Abs

Synopsis: Calculate absolute value.

Declaration: `function Abs(l: LongInt) : LongInt`
`function Abs(l: Int64) : Int64`
`function Abs(d: ValReal) : ValReal`

Visibility: default

Description: Abs returns the absolute value of a variable. The result of the function has the same type as its argument, which can be any numerical type.

Errors: None.

See also: Round ([1529](#))

Listing: ./examples/refex/ex1.pp

Program Example1;

{ Program to demonstrate the Abs function. }

Var

 r : real;
 i : integer;

begin

 r:=abs(-1.0); { r:=1.0 }
 i:=abs(-21); { i:=21 }

end.

75.12.2 AbstractError

Synopsis: Generate an abstract error.

Declaration: `procedure AbstractError`

Visibility: default

Description: `AbstractError` generates an abstract error (run-time error 211). If the `AbstractErrorProc` (1341) constant is set, it will be called instead.

Errors: This routine causes a run-time error 211.

See also: `AbstractErrorProc` (1341)

75.12.3 AcquireExceptionObject

Synopsis: Obtain a reference to the current exception object.

Declaration: `function AcquireExceptionObject : Pointer`

Visibility: default

Description: `AcquireExceptionObject` returns the current exception object. It raises the reference count of the exception object, so it will not be freed. Calling this method is only valid within an except block.

The effect of this function is countered by re-raising an exception via `raise`;

To make sure that the exception object is released when it is no longer needed, `ReleaseExceptionObject` (1523) must be called when the reference is no longer needed.

Errors: If there is no current exception, a run-time error 231 will occur.

See also: `ReleaseExceptionObject` (1523)

75.12.4 add(variant,variant):variant

Synopsis: Implement addition (+) operation on variants.

Declaration: `operator +(const op1: variant; const op2: variant) : variant`

Visibility: default

Description: The implementation of the addition + operation is delegated to the variant manager with operation `opadd`.

Errors: Execution of this operator may result in an exception if no variant manager is installed or if the types of the operand are not suitable for the operation.

See also: operator `-(variant, variant): variant` (1335)

75.12.5 AddExitProc

Synopsis: Add an exit procedure to the exit procedure chain.

Declaration: `procedure AddExitProc(Proc: TProcedure)`

Visibility: default

Description: `AddExitProc` adds `Proc` to the exit procedure chain. At program exit, all procedures added in this way will be called in reverse order.

Errors: None.

See also: `ExitProc` ([1345](#))

75.12.6 Addr

Synopsis: Return address of a variable.

Declaration: `function Addr(X: TAnytype) : Pointer`

Visibility: default

Description: `Addr` returns a pointer to its argument, which can be any type, or a function or procedure name. The returned pointer isn't typed. The same result can be obtained by the `@` operator, which can return a typed pointer (see the programmer's guide).

Errors: None

See also: `SizeOf` ([1546](#))

Listing: `./examples/refex/ex2.pp`

Program Example2;

{ Program to demonstrate the Addr function. }

Const Zero : integer = 0;

Var p : pointer;
i : Integer;

begin

p:=**Addr**(p); *{ P points to itself }*

p:=**Addr**(I); *{ P points to I }*

p:=**Addr**(Zero); *{ P points to 'Zero' }*

end.

75.12.7 Align

Synopsis: Return aligned version of an address.

Declaration: `function Align(Addr: PtrUInt; Alignment: PtrUInt) : PtrUInt`
`function Align(Addr: Pointer; Alignment: PtrUInt) : Pointer`

Visibility: default

Description: `Align` returns `Address`, aligned to `Alignment` bytes.

Errors: None.

75.12.8 AllocMem

Synopsis: Allocate and clear memory.

Declaration: `function AllocMem(Size: PtrUInt) : pointer`

Visibility: default

Description: `AllocMem` calls `getmem GetMem` ([1474](#)), and clears the allocated memory, i.e. the allocated memory is filled with `Size` zero bytes.

See also: `GetMem` ([1474](#))

75.12.9 AnsiToUtf8

Synopsis: Convert ansi string to UTF-8 string.

Declaration: `function AnsiToUtf8(const s: RawByteString) : RawByteString`

Visibility: default

Description: `AnsiToUtf8` converts the ansistring `S` to a UTF-8 format, that is, it converts the string from whatever codepage is currently in use, to UTF-8.

The current codepage is fetched from the system, if internationalization support is enabled. It can be UTF-8, in which case the function simply returns `S`.

Errors: None.

See also: `Utf8toAnsi` ([1569](#))

75.12.10 Append

Synopsis: Open a file in append mode.

Declaration: `procedure Append(var t: Text)`

Visibility: default

Description: `Append` opens an existing file in append mode. Any data written to `F` will be appended to the file. Only text files can be opened in append mode. After a call to `Append`, the file `F` becomes write-only. File sharing is not taken into account when calling `Append`.

Errors: If the file doesn't exist when appending, a run-time error will be generated. This behaviour has changed on Windows and Linux platforms, where in versions prior to 1.0.6, the file would be created in append mode.

See also: `Rewrite` ([1525](#)), `Close` ([1437](#)), `Reset` ([1524](#))

Listing: `./examples/refex/ex3.pp`

Program Example3;

{ Program to demonstrate the Append function. }

Var f : text;

begin

 Assign (f,'test.txt');

Rewrite (f); *{ file is opened for write, and emptied }*

```

Writeln (F,'This is the first line of text.txt');
close (f);
Append(f); { file is opened for write, but NOT emptied.
               any text written to it is appended.}
Writeln (f,'This is the second line of text.txt');
close (f);
end.

```

75.12.11 ArcTan

Synopsis: Calculate inverse tangent.

Declaration: `function ArcTan(d: ValReal) : ValReal`

Visibility: default

Description: `ArcTan` returns the Arctangent of X, which can be any Real type. The resulting angle is in radial units.

Errors: None

See also: `Sin` ([1545](#)), `Cos` ([1446](#))

Listing: `./examples/refex/ex4.pp`

Program Example4;

{ Program to demonstrate the ArcTan function. }

Var R : Real;

begin

 R:=**ArcTan**(0); { *R:=0* }

 R:=**ArcTan**(1)/pi; { *R:=0.25* }

end.

75.12.12 ArrayStringToPPchar

Synopsis: Convert an array of string to an array of null-terminated strings.

Declaration: `function ArrayStringToPPchar(const S: Array of AnsiString;
 reserveentries: LongInt) : PPChar`

Visibility: default

Description: `ArrayStringToPPchar` creates an array of null-terminated strings that point to strings which are the same as the strings in the array S. The function returns a pointer to this array. The array and the strings it contains must be disposed of after being used, because it they are allocated on the heap.

The `ReserveEntries` parameter tells `ArrayStringToPPchar` to allocate room at the end of the array for another `ReserveEntries` entries.

Errors: If not enough memory is available, an error may occur.

See also: `StringToPPChar` ([1552](#))

75.12.13 Assert

Synopsis: Check validity of a given condition.

Declaration: `procedure Assert (Expr: Boolean)`
`procedure Assert (Expr: Boolean; const Msg: string)`

Visibility: default

Description: With assertions on, `Assert` tests if `expr` is false, and if so, aborts the application with a Runtime error 227 and an optional error message in `msg`. If `expr` is true, program execution continues normally. If assertions are not enabled at compile time, this routine does nothing, and no code is generated for the `Assert` call. Enabling and disabling assertions at compile time is done via the `\$C` or `\$ASSERTIONS` compiler switches. These are local switches. The default behavior of the `assert` call can be changed by setting a new handler in the `AssertErrorProc` variable. `Sysutils` overrides the default handler to raise a `EAssertionFailed` exception.

Errors: None.

See also: `Halt` ([1481](#)), `Runerror` ([1532](#))

75.12.14 Assign

Synopsis: Assign a name to a file.

Declaration: `procedure Assign(out f: File; const Name: ShortString)`
`procedure Assign(out f: File; const p: PAnsiChar)`
`procedure Assign(out f: File; const c: AnsiChar)`
`procedure Assign(out f: File; const Name: UnicodeString)`
`procedure Assign(out f: File; const Name: RawByteString)`
`procedure Assign(out f: TypedFile; const Name: shortstring)`
`procedure Assign(out f: TypedFile; const p: PAnsiChar)`
`procedure Assign(out f: TypedFile; const c: AnsiChar)`
`procedure Assign(out f: TypedFile; const Name: unicodestring)`
`procedure Assign(out f: TypedFile; const Name: RawByteString)`
`procedure Assign(out t: Text; const s: shortstring)`
`procedure Assign(out t: Text; const p: PAnsiChar)`
`procedure Assign(out t: Text; const c: AnsiChar)`
`procedure Assign(out t: Text; const s: unicodestring)`
`procedure Assign(out t: Text; const s: RawByteString)`

Visibility: default

Description: `Assign` assigns a name to `F`, which can be any file type. This call doesn't open the file, it just assigns a name to a file variable, and marks the file as closed.

Note that the filename (including path) can be only 255 characters long.

Errors: None.

See also: `Reset` ([1524](#)), `Rewrite` ([1525](#)), `Append` ([1421](#))

Listing: `./examples/refex/ex5.pp`

Program Example5;

{ Program to demonstrate the Assign function. }

Var F : text;


```

begin
  Assign (F,"");
  Rewrite (f);
  { The following can be put in any file by redirecting it
    from the command line.}
  Writeln (f,'This goes to standard output !');
  Close (f);
  Assign (F,'Test.txt');
  rewrite (f);
  writeln (f,'This doesn"t go to standard output !');
  close (f);
end.

```

75.12.15 assign(Comp):olevariant

Synopsis: Assign a comp-precision float to an ole-variant.

Declaration: `operator :=(const source: Comp) : olevariant`

Visibility: default

75.12.16 assign(Comp):variant

Synopsis: Assign a comp-precision float to a variant.

Declaration: `operator :=(const source: Comp) : variant`

Visibility: default

Description: The resulting variant is a double-precision value.

75.12.17 assign(extended):olevariant

Synopsis: Assign an extended-precision float to an ole-variant.

Declaration: `operator :=(const source: extended) : olevariant`

Visibility: default

75.12.18 assign(extended):variant

Synopsis: Assign an extended-precision float to a variant.

Declaration: `operator :=(const source: extended) : variant`

Visibility: default

Description: The resulting variant is a double-precision value

75.12.19 assign(NativeInt):variant

Declaration: `operator :=(const source: NativeInt) : variant`

Visibility: default

75.12.20 **assign(NativeUInt):variant**

Declaration: `operator :=(const source: NativeUInt) : variant`

Visibility: default

75.12.21 **assign(olevariant):Comp**

Synopsis: Assign an ole- variant to a comp-precision float.

Declaration: `operator :=(const source: olevariant) : Comp`

Visibility: default

75.12.22 **assign(olevariant):extended**

Synopsis: Assign an ole- variant to an extended-precision float.

Declaration: `operator :=(const source: olevariant) : extended`

Visibility: default

75.12.23 **assign(olevariant):Real**

Synopsis: Assign an ole- variant to a real-precision float.

Declaration: `operator :=(const source: olevariant) : Real`

Visibility: default

75.12.24 **assign(olevariant):single**

Synopsis: Assign an ole- variant to a single-precision float.

Declaration: `operator :=(const source: olevariant) : single`

Visibility: default

75.12.25 **assign(olevariant):UnicodeString**

Synopsis: Assign an ole-variant to a unicode string.

Declaration: `operator :=(const source: olevariant) : UnicodeString`

Visibility: default

75.12.26 **assign(Real):olevariant**

Synopsis: Assign a real-precision float to an ole-variant.

Declaration: `operator :=(const source: Real) : olevariant`

Visibility: default

75.12.27 assign(Real):variant

Synopsis: Assign a real-precision float to a variant.

Declaration: `operator :=(const source: Real) : variant`

Visibility: default

Description: The resulting variant is a double-precision value.

75.12.28 assign(Real48):extended

Synopsis:

Declaration: `operator :=(b: Real48) : extended`

Visibility: default

75.12.29 assign(single):olevariant

Synopsis: Assign a single-precision float to an ole-variant.

Declaration: `operator :=(const source: single) : olevariant`

Visibility: default

75.12.30 assign(single):variant

Synopsis: Assign a single-precision float to a variant.

Declaration: `operator :=(const source: single) : variant`

Visibility: default

Description: The resulting variant is a double-precision value

75.12.31 assign(UCS4String):variant

Synopsis: Assign UCS4String to a variant, performing the necessary conversions.

Declaration: `operator :=(const source: UCS4String) : variant`

Visibility: default

Description: The resulting variant is a widestring.

75.12.32 assign(UnicodeString):olevariant

Synopsis: Assign a unicodestring to an ole-variant.

Declaration: `operator :=(const source: UnicodeString) : olevariant`

Visibility: default

75.12.33 assign(UnicodeString):variant

Synopsis: Assign UnicodeString to a variant, performing the necessary conversions.

Declaration: `operator :=(const source: UnicodeString) : variant`

Visibility: default

Description: The resulting variant is a widestring.

75.12.34 assign(UTF8String):variant

Synopsis: Assign an UTF-8 string to a variant, performing the necessary conversions.

Declaration: `operator :=(const source: UTF8String) : variant`

Visibility: default

Description: The resulting variant is a widestring.

75.12.35 assign(variant):Comp

Synopsis: Assign a variant to a comp-precision float.

Declaration: `operator :=(const source: variant) : Comp`

Visibility: default

75.12.36 assign(variant):extended

Synopsis: Assign a variant to a extended-precision float.

Declaration: `operator :=(const source: variant) : extended`

Visibility: default

75.12.37 assign(variant):NativeInt

Declaration: `operator :=(const source: variant) : NativeInt`

Visibility: default

75.12.38 assign(variant):NativeUInt

Declaration: `operator :=(const source: variant) : NativeUInt`

Visibility: default

75.12.39 assign(variant):Real

Synopsis: Assign a variant to a real-precision float.

Declaration: `operator :=(const source: variant) : Real`

Visibility: default

75.12.40 assign(variant):single

Synopsis: Assign a variant to a single-precision float.

Declaration: `operator :=(const source: variant) : single`

Visibility: default

75.12.41 assign(variant):unicodestring

Synopsis: Assign a variant to a unicodestring.

Declaration: `operator :=(const source: variant) : unicodestring`

Visibility: default

75.12.42 assign(variant):UTF8String

Synopsis: Assign a variant to an UTF8String.

Declaration: `operator :=(const source: variant) : UTF8String`

Visibility: default

75.12.43 Assigned

Synopsis: Check if a pointer is valid.

Declaration: `function Assigned(P: Pointer) : Boolean`

Visibility: default

Description: `Assigned` returns `True` if `P` is non-nil and returns `False` if `P` is nil. The main use of `Assigned` is that Procedural variables, method variables and class-type variables also can be passed to `Assigned`.

Errors: None

See also: [New \(1509\)](#)

Listing: `./examples/refex/ex96.pp`

Program Example96;

{ Program to demonstrate the Assigned function. }

Var P : Pointer;

begin

If Not Assigned(P) **then**

Writeln ('Pointer is initially NIL');

 P:=@P;

If Not Assigned(P) **then**

Writeln('Internal inconsistency')

else

Writeln('All is well in FPC')

end.

75.12.44 BasicEventCreate

Synopsis: Obsolete. Don't use.

Declaration: `function BasicEventCreate(EventAttributes: Pointer;
 AManualReset: Boolean; InitialState: Boolean;
 const Name: ansistring) : PEventState`

Visibility: default

Description: `BasicEventCreate` is obsolete, use `RTLEventCreate` ([1531](#)) instead.

See also: `RTLEventCreate` ([1531](#))

75.12.45 BasicEventDestroy

Synopsis: Obsolete. Don't use.

Declaration: `procedure BasicEventDestroy(state: PEventState)`

Visibility: default

Description: `basiceventdestroy` is obsolete. Use `RTLEventDestroy` ([1531](#)) instead.

See also: `RTLEventDestroy` ([1531](#))

75.12.46 BasicEventResetEvent

Synopsis: Obsolete. Don't use.

Declaration: `procedure BasicEventResetEvent(state: PEventState)`

Visibility: default

Description: `basiceventresetevent` is obsolete. Use `RTLEventResetEvent` ([1531](#)) instead.

See also: `RTLEventResetEvent` ([1531](#))

75.12.47 BasicEventSetEvent

Synopsis: Obsolete. Don't use.

Declaration: `procedure BasicEventSetEvent(state: PEventState)`

Visibility: default

Description: `basiceventsetevent` is obsolete. Use `RTLEventSetEvent` ([1532](#)) instead.

See also: `RTLEventSetEvent` ([1532](#))

75.12.48 BasicEventWaitFor

Synopsis: Obsolete. Don't use.

Declaration: `function BasicEventWaitFor(Timeout: Cardinal; state: PEventState;
 FUseComWait: Boolean) : LongInt`

Visibility: default

Description: `basiceventwaitfor` is obsolete. Use `RTLEventWaitFor` ([1532](#)) instead.

See also: `RTLEventWaitFor` ([1532](#))

75.12.49 BeginThread

Synopsis: Start a new thread.

Declaration: `function BeginThread(sa: Pointer; stacksize: SizeUInt; ThreadFunction: TThreadFunc; p: pointer; creationFlags: DWord; var ThreadId: TThreadID) : TThreadID`

`function BeginThread(ThreadFunction: TThreadFunc) : TThreadID`

`function BeginThread(ThreadFunction: TThreadFunc; p: pointer) : TThreadID`

`function BeginThread(ThreadFunction: TThreadFunc; p: pointer; var ThreadId: TThreadID) : TThreadID`

`function BeginThread(ThreadFunction: TThreadFunc; p: pointer; var ThreadId: TThreadID; const stacksize: SizeUInt) : TThreadID`

Visibility: default

Description: `BeginThread` starts a new thread and executes `ThreadFunction` in the new thread. If `P` is specified, then it is passed to `ThreadFunction`. If `ThreadId` is specified, it is filled with the thread ID of the newly started thread. If `StackSize` is specified, it is set as the stack size for the new thread. If none is specified, a default stack size of 4MiB is used.

The function returns the thread handle (or ID, on some other operating systems like Linux or OS/2) on success, or 0 if an error occurred. Note that the thread ID and handle are the same on Unix processes, and that the thread ID and thread handle are different on windows systems.

Errors: On error, the value "0" is returned.

See also: `EndThread` ([1455](#))

75.12.50 BEtoN

Synopsis: Convert Big Endian-ordered integer to Native-ordered integer.

Declaration: `function BEtoN(const AValue: SmallInt) : SmallInt`

`function BEtoN(const AValue: Word) : Word`

`function BEtoN(const AValue: LongInt) : LongInt`

`function BEtoN(const AValue: DWord) : DWord`

`function BEtoN(const AValue: Int64) : Int64`

`function BEtoN(const AValue: QWord) : QWord`

Visibility: default

Description: `BEtoN` will rearrange the bytes in a Big-Endian number to the native order for the current processor. That is, for a big-endian processor, it will do nothing, and for a little-endian processor, it will invert the order of the bytes.

See also: `LEtoN` ([1501](#)), `NtoBE` ([1510](#)), `NtoLE` ([1510](#))

75.12.51 BinStr

Synopsis: Convert integer to string with binary representation.

Declaration: `function BinStr(Val: LongInt; cnt: Byte) : shortstring`

`function BinStr(Val: Int64; cnt: Byte) : shortstring`

`function BinStr(Val: QWord; cnt: Byte) : shortstring`

Visibility: default

Description: `BinStr` returns a string with the binary representation of `Value`. The string has at most `cnt` characters. (i.e. only the `cnt` rightmost bits are taken into account) To have a complete representation of any longint-type value, 32 bits are needed, i.e. `cnt=32`

Errors: None.

See also: `Str` ([1549](#)), `Val` ([1570](#)), `HexStr` ([1481](#)), `OctStr` ([1511](#))

Listing: `./examples/refex/ex82.pp`

```

Program example82;

{ Program to demonstrate the BinStr function }

Const Value = 45678;

Var I : longint;

begin
  For I:=8 to 20 do
    WriteLn (BinStr(Value,I):20);
end.

```

75.12.52 BlockRead

Synopsis: Read data from an untyped file into memory.

Declaration:

```

procedure BlockRead(var f: File; var Buf; count: Int64;
                    var Result: Int64)
procedure BlockRead(var f: File; var Buf; count: LongInt;
                    var Result: LongInt)
procedure BlockRead(var f: File; var Buf; count: Cardinal;
                    var Result: Cardinal)
procedure BlockRead(var f: File; var Buf; count: Word; var Result: Word)
procedure BlockRead(var f: File; var Buf; count: Word;
                    var Result: Integer)
procedure BlockRead(var f: File; var Buf; count: Int64)

```

Visibility: default

Description: `Blockread` reads `count` or less records from file `F`. A record is a block of bytes with size specified by the `Rewrite` ([1525](#)) or `Reset` ([1524](#)) statement. The result is placed in `Buffer`, which must contain enough room for `Count` records. The function cannot read partial records. If `Result` is specified, it contains the number of records actually read. If `Result` isn't specified, and less than `Count` records were read, a run-time error is generated. This behavior can be controlled by the `{SI}` switch.

Errors: Depending on the state of the `{SI}` switch, a runtime error can be generated if there is an error. In the `{SI-}` state, use `IOResult` to check for errors.

See also: `Blockwrite` ([1432](#)), `Close` ([1437](#)), `Reset` ([1524](#)), `Assign` ([1423](#))

Listing: `./examples/refex/ex6.pp`

Program Example6;

{ Program to demonstrate the BlockRead and BlockWrite functions. }

```

Var Fin, fout : File;
      NumRead, NumWritten : Word;
      Buf : Array[1..2048] of byte;
      Total : Longint;

begin
  Assign (Fin, Paramstr(1));
  Assign (Fout, Paramstr(2));
  Reset (Fin, 1);
  Rewrite (Fout, 1);
  Total := 0;
  Repeat
    BlockRead (Fin, buf, Sizeof(buf), NumRead);
    BlockWrite (Fout, Buf, NumRead, NumWritten);
    inc(Total, NumWritten);
  Until (NumRead = 0) or (NumWritten <> NumRead);
  Write ('Copied ', Total, ' bytes from file ', paramstr(1));
  Writeln (' to file ', paramstr(2));
  close(fin);
  close(fout);
end.

```

75.12.53 BlockWrite

Synopsis: Write data from memory to an untyped file.

Declaration: `procedure BlockWrite(var f: File; const Buf; Count: Int64; var Result: Int64)`
`procedure BlockWrite(var f: File; const Buf; Count: LongInt; var Result: LongInt)`
`procedure BlockWrite(var f: File; const Buf; Count: Cardinal; var Result: Cardinal)`
`procedure BlockWrite(var f: File; const Buf; Count: Word; var Result: Word)`
`procedure BlockWrite(var f: File; const Buf; Count: Word; var Result: Integer)`
`procedure BlockWrite(var f: File; const Buf; Count: LongInt)`

Visibility: default

Description: `BlockWrite` writes count records from buffer to the file F. A record is a block of bytes with size specified by the `Rewrite` (1525) or `Reset` (1524) statement. If the records couldn't be written to disk, a run-time error is generated. This behavior can be controlled by the `{SI}` switch.

Errors: Depending on the state of the `{SI}` switch, a runtime error can be generated if there is an error. In the `{SI-}` state, use `IOResult` to check for errors.

See also: `Blockread` (1431), `Close` (1437), `Rewrite` (1525), `Assign` (1423)

75.12.54 Break

Synopsis: Exit current loop construct.

Declaration: `procedure Break`

Visibility: `default`

Description: `Break` jumps to the statement following the end of the current repetitive statement. The code between the `Break` call and the end of the repetitive statement is skipped. The condition of the repetitive statement is NOT evaluated.

This can be used with `For`, `repeat` and `While` statements.

Note that although `Break` is a compiler intrinsic (i.e. is treated specially) it is defined as a procedure in the system unit, and hence can be redefined.

Errors: None.

See also: `Continue` ([1444](#)), `Exit` ([1460](#))

Listing: `./examples/refex/ex87.pp`

Program `Example87;`

{ Program to demonstrate the Break function. }

Var `l : longint;`

begin

`l:=0;`

While `l<10 Do`

begin

`Inc(l);`

If `l>5 Then`

Break;

`Writeln (i);`

end;

`l:=0;`

Repeat

`Inc(l);`

If `l>5 Then`

Break;

`Writeln (i);`

Until `l>=10;`

For `l:=1 to 10 do`

begin

If `l>5 Then`

Break;

`Writeln (i);`

end;

end.

Listing: `./examples/refex/ex121.pp`

{

Example 121:

Continue, break and exit are system procedures.

They can be redefined

}

procedure `continue;`

begin

```

    Writeln('Continue');
end;

```

```

Procedure Exit;

```

```

begin
    Writeln('exit');
end;

```

```

Procedure Break;

```

```

begin
    Writeln('Break');
end;

```

```

begin
    Repeat
        Continue;
        Break;
    until exit;
end.

```

75.12.55 BsfByte

Synopsis: Return the position of the rightmost set bit in an 8-bit value.

Declaration: `function BsfByte(const AValue: Byte) : Byte`

Visibility: default

Description: `BsfByte` scans the byte `AValue`, starting at position 0 (rightmost position) and returns the index of the first set bit. The position is measured from the 0-th, rightmost bit.

When the input is 0, the result is 255 (unsigned equivalent of -1).

See also: `BsrByte` ([1435](#)), `BsfWord` ([1435](#)), `BsfDWord` ([1434](#)), `BsfQWord` ([1434](#))

75.12.56 BsfDWord

Synopsis: Return the position of the rightmost set bit in a 32-bit value.

Declaration: `function BsfDWord(const AValue: DWord) : Cardinal`

Visibility: default

Description: `BsfDWord` scans the `DWord` `AValue`, starting at position 0 (rightmost position) , and returns the index of the first set bit. The position is measured from the 0-th, rightmost bit.

When the input is 0, the result is 255 (unsigned equivalent of -1).

See also: `BsfByte` ([1434](#)), `BsfWord` ([1435](#)), `BsrDWord` ([1435](#)), `BsfQWord` ([1434](#))

75.12.57 BsfQWord

Synopsis: Return the position of the rightmost set bit in a 64-bit value.

Declaration: `function BsfQWord(const AValue: QWord) : Cardinal`

Visibility: default

Description: `BsfQWord` scans the `QWord AValue`, starting at position 0 (rightmost position) , and returns the index of the first set bit. The position is measured from the 0-th, rightmost bit.

When the input is 0, the result is 255 (unsigned equivalent of -1).

See also: `BsfByte` ([1434](#)), `BsfWord` ([1435](#)), `BsfDWord` ([1434](#)), `BsrQWord` ([1436](#))

75.12.58 BsfWord

Synopsis: Return the position of the rightmost set bit in a 16-bit value.

Declaration: `function BsfWord(const AValue: Word) : Cardinal`

Visibility: default

Description: `BsfWord` scans the word `AValue`, starting at position 0 (rightmost position) , and returns the index of the first set bit. The position is measured from the 0-th, rightmost bit.

When the input is 0, the result is 255 (unsigned equivalent of -1).

See also: `BsfByte` ([1434](#)), `BsrWord` ([1436](#)), `BsfDWord` ([1434](#)), `BsfQWord` ([1434](#))

75.12.59 BsrByte

Synopsis: Return the position of the leftmost set bit in an 8-bit value.

Declaration: `function BsrByte(const AValue: Byte) : Byte`

Visibility: default

Description: `BsfByte` scans the byte `AValue`, starting at the leftmost position and working towards position 0, and returns the index of the first set bit. The position is measured from the 0-th, rightmost bit.

When the input is 0, the result is 255 (unsigned equivalent of -1).

See also: `BsfByte` ([1434](#)), `BsrWord` ([1436](#)), `BsrDWord` ([1435](#)), `BsrQWord` ([1436](#))

75.12.60 BsrDWord

Synopsis: Return the position of the leftmost set bit in a 32-bit value.

Declaration: `function BsrDWord(const AValue: DWord) : Cardinal`

Visibility: default

Description: `BsrDWord` scans the `DWord AValue`, starting at the leftmost position and working towards position 0, and returns the index of the first set bit. The position is measured from the 0-th, rightmost bit.

When the input is 0, the result is 255 (unsigned equivalent of -1).

See also: `BsrByte` ([1435](#)), `BsrWord` ([1436](#)), `BsfDWord` ([1434](#)), `BsrQWord` ([1436](#))

75.12.61 BsrQWord

Synopsis: Return the position of the leftmost set bit in a 64-bit value.

Declaration: `function BsrQWord(const AValue: QWord) : Cardinal`

Visibility: default

Description: `BsfQWord` scans the `QWord` `AValue`, starting at the leftmost position and working towards position 0, and returns the index of the first set bit. The position is measured from the 0-th, rightmost bit.

When the input is 0, the result is 255 (unsigned equivalent of -1).

See also: `BsfByte` ([1434](#)), `BsfWord` ([1435](#)), `BsfDWord` ([1434](#)), `BsrQWord` ([1436](#))

75.12.62 BsrWord

Synopsis: Return the position of the leftmost set bit in a 16-bit value.

Declaration: `function BsrWord(const AValue: Word) : Cardinal`

Visibility: default

Description: `BsrWord` scans the word `AValue`, starting at the leftmost position and working towards position 0, and returns the index of the first set bit. The position is measured from the 0-th, rightmost bit.

When the input is 0, the result is 255 (unsigned equivalent of -1).

See also: `BsrByte` ([1435](#)), `BsfWord` ([1435](#)), `BsrDWord` ([1435](#)), `BsrQWord` ([1436](#))

75.12.63 CaptureBacktrace

Synopsis: Return stack trace.

Declaration: `function CaptureBacktrace(skipframes: SizeInt; count: SizeInt; frames: PCodePointer) : SizeInt`

Visibility: default

Description: `CaptureBacktrace` will fill the array pointed to by `frames` with the addresses of a backtrace. It will skip `skipframes` frames, and will write at most `count` addresses. Frames must point to enough memory to hold the stacktrace, which is `count*sizeof(codepointer)` bytes.

See also: `Get_pc_addr` ([1480](#)), `get_caller_stackinfo` ([1479](#)), `get_caller_addr` ([1479](#)), `get_caller_frame` ([1479](#))

75.12.64 ChDir

Synopsis: Change current working directory.

Declaration: `procedure ChDir(const s: shortstring); Overload`
`procedure ChDir(const s: RawByteString); Overload`
`procedure ChDir(const s: unicodestring); Overload`

Visibility: default

Description: `Chdir` changes the working directory of the process to `S`.

Errors: Depending on the state of the `{SI}` switch, a runtime error can be generated if there is an error. In the `{SI-}` state, use `IOResult` to check for errors.

See also: Mkdir ([1507](#)), Rmdir ([1526](#))

Listing: ./examples/refex/ex7.pp

Program Example7;

{ Program to demonstrate the ChDir function. }

```
begin
  {$I-}
  ChDir (ParamStr(1));
  if IOresult<>0 then
    Writeln ('Cannot change to directory : ',paramstr (1));
end.
```

75.12.65 Chr

Synopsis: Convert byte value to character value.

Declaration: `function Chr(b: Byte) : Char`

Visibility: default

Description: Chr returns the character which has ASCII value X.

Historical note:

Originally, Pascal did not have typecasts and chr was a necessary function in order to do certain operations on ASCII values of characters. With the arrival of typecasting a generic approach became possible, making chr mostly obsolete. However, chr is not considered deprecated and remains in wide use today.

Errors: None.

See also: Ord ([1512](#)), Str ([1549](#))

Listing: ./examples/refex/ex8.pp

Program Example8;

{ Program to demonstrate the Chr function. }

```
begin
  Write (chr(10),chr(13)); { The same effect as Writeln; }
end.
```

75.12.66 Close

Synopsis: Close a file.

Declaration: `procedure Close(var f: File)`
`procedure Close(var t: Text)`

Visibility: default

Description: Close flushes the buffer of the file F and closes F. After a call to Close, data can no longer be read from or written to F. To reopen a file closed with Close, it isn't necessary to assign the file again. A call to Reset ([1524](#)) or Rewrite ([1525](#)) is sufficient.

Errors: None.

See also: [CompareChar \(1439\)](#), [CompareChar0 \(1441\)](#), [CompareWord \(1442\)](#), [CompareDWord \(1441\)](#)

Listing: ./examples/refex/ex99.pp

Program Example99;

{ Program to demonstrate the CompareByte function. }

Const

 ArraySize = 100;
 HalfArraySize = ArraySize **Div** 2;

Var

 Buf1, Buf2 : **Array**[1..ArraySize] **of** byte;
 I : longint;

Procedure CheckPos(Len : Longint);

Begin

Write('First ', Len, ' positions are ');
 if CompareByte(Buf1, Buf2, Len) <> 0 **then**
 Write('NOT ');
 Writeln('equal');
 end;

begin

For I:=1 **to** ArraySize **do**
 begin
 Buf1[I]:=I;
 If I<=HalfArraySize **Then**
 Buf2[I]:=I
 else
 Buf2[I]:=HalfArraySize-I;
 end;
 CheckPos(HalfArraySize **div** 2);
 CheckPos(HalfArraySize);
 CheckPos(HalfArraySize+1);
 CheckPos(HalfArraySize + HalfArraySize **Div** 2);
 end.

75.12.69 CompareChar

Synopsis: compare 2 memory buffers character per character.

Declaration: `function CompareChar(const buf1; const buf2; len: SizeInt) : SizeInt`

Visibility: default

Description: `CompareChar` compares two memory regions `buf1`, `buf2` on a character-per-character basis for a total of `len` characters.

The `CompareChar0` variant compares `len` bytes, or until a zero character is found.

The function returns one of the following values:

- < 0 if buf1 and buf2 contain different characters in the first len positions, and the first such character is smaller in buf1 than the character at the same position in buf2.
- 0 if the first len characters in buf1 and buf2 are equal.
- > 0 if buf1 and buf2 contain different characters in the first len positions, and the first such character is larger in buf1 than the character at the same position in buf2.

Errors: None.

See also: CompareByte ([1438](#)), CompareChar0 ([1441](#)), CompareWord ([1442](#)), CompareDWord ([1441](#))

Listing: ./examples/refex/ex100.pp

Program Example100;

{ Program to demonstrate the CompareChar function. }

Const

 ArraySize = 100;
 HalfArraySize = ArraySize **Div** 2;

Var

 Buf1, Buf2 : **Array**[1..ArraySize] **of** char;
 I : longint;

Procedure CheckPos(Len : Longint);

Begin

Write('First ', Len, ' characters are ');
 if CompareChar(Buf1, Buf2, Len) <> 0 **then**
 Write('NOT ');
 Writeln('equal');
end;

Procedure CheckNullPos(Len : Longint);

Begin

Write('First ', Len, ' non-null characters are ');
 if CompareChar0(Buf1, Buf2, Len) <> 0 **then**
 Write('NOT ');
 Writeln('equal');
end;

begin

For I:=1 **to** ArraySize **do**
 begin
 Buf1[I]:=chr(I);
 if I<=HalfArraySize **Then**
 Buf2[I]:=chr(I)
 else
 Buf2[I]:=chr(HalfArraySize-I);
 end;
 CheckPos(HalfArraySize **div** 2);
 CheckPos(HalfArraySize);
 CheckPos(HalfArraySize+1);
 CheckPos(HalfArraySize + HalfArraySize **Div** 2);
 For I:=1 **to** 4 **do**
 begin
 buf1[**Random**(ArraySize)+1]:=Chr(0);

```

    buf2[Random(ArraySize)+1]:=Chr(0);
  end;
  Randomize;
  CheckNullPos(HalfArraySize div 2);
  CheckNullPos(HalfArraySize);
  CheckNullPos(HalfArraySize+1);
  CheckNullPos(HalfArraySize + HalfArraySize Div 2);
end.

```

75.12.70 CompareChar0

Synopsis: Compare two buffers character by character till a null-character is reached.

Declaration: `function CompareChar0(const buf1; const buf2; len: SizeInt) : SizeInt`

Visibility: default

Description: `CompareChar0` compares 2 buffers `buf1` and `buf2` for a maximum length of `len` or till a null character is reached in either buffer. The result depends on the contents of the buffers:

- < 0 If `buf1` contains a character less than the corresponding character in `buf2`.
- 0 If both buffers are equal
- > 0 If `buf1` contains a character greater than the corresponding character in `buf2`.

Errors: None.

See also: `CompareByte` ([1438](#)), `CompareChar` ([1439](#)), `CompareDWord` ([1441](#)), `CompareWord` ([1442](#))

75.12.71 CompareDWord

Synopsis: Compare 2 memory buffers DWord per DWord.

Declaration: `function CompareDWord(const buf1; const buf2; len: SizeInt) : SizeInt`

Visibility: default

Description: `CompareDWord` compares two memory regions `buf1`, `buf2` on a DWord-per-DWord basis for a total of `len` DWords. (A DWord is 4 bytes).

The function returns one of the following values:

- < 0 if `buf1` and `buf2` contain different DWords in the first `len` DWords, and the first such DWord is smaller in `buf1` than the DWord at the same position in `buf2`.
- 0 if the first `len` DWords in `buf1` and `buf2` are equal.
- > 0 if `buf1` and `buf2` contain different DWords in the first `len` DWords, and the first such DWord is larger in `buf1` than the DWord at the same position in `buf2`.

Errors: None.

See also: `CompareChar` ([1439](#)), `CompareByte` ([1438](#)), `CompareWord` ([1442](#))

Listing: `./examples/refex/ex101.pp`

Program Example101;

{ Program to demonstrate the CompareDWord function. }

Const
 ArraySize = 100;
 HalfArraySize = ArraySize **Div** 2;

Var
 Buf1, Buf2 : **Array**[1..ArraySize] **of** Dword;
 I : longint;

Procedure CheckPos(Len : Longint);

Begin
 Write('First ', Len, ' DWords are ');
 if CompareDWord(Buf1, Buf2, Len) <> 0 then
 Write('NOT ');
 Writeln('equal');
 end;

begin
 For I:=1 to ArraySize do
 begin
 Buf1[I]:=I;
 If I<=HalfArraySize **Then**
 Buf2[I]:=I
 else
 Buf2[I]:=HalfArraySize-I;
 end;
 CheckPos(HalfArraySize **div** 2);
 CheckPos(HalfArraySize);
 CheckPos(HalfArraySize+1);
 CheckPos(HalfArraySize + HalfArraySize **Div** 2);
 end.

75.12.72 CompareWord

Synopsis: Compare 2 memory buffers word per word.

Declaration: function CompareWord(const buf1; const buf2; len: SizeInt) : SizeInt

Visibility: default

Description: CompareWord compares two memory regions buf1, buf2 on a Word-per-Word basis for a total of len Words. (A Word is 2 bytes).

The function returns one of the following values:

- < 0 if buf1 and buf2 contain different Words in the first len Words, and the first such Word is smaller in buf1 than the Word at the same position in buf2.
- 0 if the first len Words in buf1 and buf2 are equal.
- > 0 if buf1 and buf2 contain different Words in the first len Words, and the first such Word is larger in buf1 than the Word at the same position in buf2.

Errors: None.

See also: `CompareChar` ([1439](#)), `CompareByte` ([1438](#)), `CompareDWord` ([1441](#))

Listing: ./examples/refex/ex102.pp

Program Example102;

{ Program to demonstrate the CompareWord function. }

Const

ArraySize = 100;
HalfArraySize = ArraySize **Div** 2;

Var

Buf1, Buf2 : **Array**[1..ArraySize] **of** Word;
I : longint;

Procedure CheckPos(Len : Longint);

Begin

Write('First ', Len, ' words are ');
if CompareWord(Buf1, Buf2, Len) <> 0 **then**
Write('NOT ');
WriteLn('equal');
end;

begin

For I:=1 **to** ArraySize **do**
begin
Buf1[I]:=I;
If I<=HalfArraySize **Then**
Buf2[I]:=I
else
Buf2[I]:=HalfArraySize-I;
end;
CheckPos(HalfArraySize **div** 2);
CheckPos(HalfArraySize);
CheckPos(HalfArraySize+1);
CheckPos(HalfArraySize + HalfArraySize **Div** 2);
end.

75.12.73 Concat

Synopsis: Append one string or dynamic array to another.

Declaration: `function Concat(const S1: string; const S2: string; const S3: string;
const Sn: string) : string`

Visibility: default

Description: `Concat` concatenates the strings `S1, S2` etc. to one long string. The same operation can be performed with the `+` operation.

`Concat` can also be used to concatenate 2 dynamic arrays of any type, resulting in a new dynamic array containing all the elements of the dynamic arrays used in the call.

Errors: None.

See also: Copy ([1445](#)), Delete ([1449](#)), Insert ([1493](#)), Pos ([1515](#)), Length ([1500](#))

Listing: ./examples/refex/ex10.pp

Program Example10;

{ Program to demonstrate the Concat function. }

Var

S : String;

begin

S:=Concat('This can be done',' Easier ','with the + operator !');

end.

75.12.74 Continue

Synopsis: Continue with next loop cycle.

Declaration: `procedure Continue`

Visibility: `default`

Description: `Continue` jumps to the end of the current repetitive statement. The code between the `Continue` call and the end of the repetitive statement is skipped. The condition of the repetitive statement is then checked again.

This can be used with `For`, `repeat` and `While` statements.

Note that although `Continue` is a compiler intrinsic (i.e. is treated specially) it is defined as an identifier in the system unit, hence it can be redefined.

Errors: None.

See also: Break ([1432](#)), Exit ([1460](#))

Listing: ./examples/refex/ex86.pp

Program Example86;

{ Program to demonstrate the Continue function. }

Var I : longint;

begin

 I:=0;

While I<10 **Do**

begin

Inc(I);

If I<5 **Then**

Continue;

Writeln (I);

end;

 I:=0;

Repeat

Inc(I);

If I<5 **Then**

Continue;

Writeln (I);

Until I>=10;

```

For I:=1 to 10 do
  begin
    If I<5 Then
      Continue;
    Writeln (i);
  end;
end.

```

Listing: ./examples/refex/ex121.pp

```

{
  Example 121:
  Continue, break and exit are system procedures.
  They can be redefined
}

```

```

procedure continue;

```

```

begin
  Writeln('Continue');
end;

```

```

Procedure Exit;

```

```

begin
  Writeln('exit');
end;

```

```

Procedure Break;

```

```

begin
  Writeln('Break');
end;

```

```

begin
  Repeat
    Continue;
    Break;
  exit;
  Until True;
end.

```

75.12.75 Copy

Synopsis: Copy part of a string or dynamic array.

Declaration: `function Copy(S: AStringType; Index: SizeInt; Count: SizeInt) : string`
`function Copy(A: DynArrayType; Index: SizeInt; Count: SizeInt)`
`: DynArray`

Visibility: default

Description: Copy returns a string which is a copy if the Count characters in S, starting at position Index. If Count is larger than the length of the string S, the result is truncated. If Index is larger than the length of the string S, then an empty string is returned. Index is 1-based.

For dynamic arrays, Copy returns a new dynamic array of the same type as the original one, and copies Count elements from the old array, starting at the position in Index. Both the Index and

Count arguments are optional: if none are specified, the whole array is copied. This is useful to make sure you have a reference count of 1 on the dynamic array (i.e. similar to UniqueString (1566)). Note that the copy is a shallow copy, i.e. sub-arrays are not copied as well.

The Count argument can be omitted. In that case, the string (or dynamic array) is copied from the position Index till the end of the string or array.

Errors: None.

See also: Delete (1449), Insert (1493), Pos (1515)

Listing: ./examples/refex/ex11.pp

Program Example11;

{ Program to demonstrate the Copy function. }

Var S,T : **String**;

begin

T:='1234567';

S:=**Copy** (T,1,2); { S:='12' }

S:=**Copy** (T,4,2); { S:='45' }

S:=**Copy** (T,4,8); { S:='4567' }

end.

75.12.76 CopyArray

Synopsis: Copy managed-type elements in array.

Declaration: `procedure CopyArray(dest: Pointer; source: Pointer; typeInfo: Pointer;
count: SizeInt)`

Visibility: default

Description: CopyArray copies count elements containing managed types from the array pointed to by source to the array pointed to by dest. For this, it uses the type information of the elements as specified in typeinfo.

Under normal circumstances, this procedure should not be used, it is called automatically by the compiler when an array-typed variables are assigned to each other.

See also: InitializeArray (1492), FinalizeArray (1467), DynArraySize (1454), DynArrayClear (1453), DynArrayDim (1453), DynArrayBounds (1453)

75.12.77 Cos

Synopsis: Calculate cosine of angle.

Declaration: `function Cos(d: ValReal) : ValReal`

Visibility: default

Description: Cos returns the cosine of X, where X is an angle, in radians. If the absolute value of the argument is larger than 263, then the result is undefined.

Errors: None.

See also: Arctan (1422), Sin (1545)

Listing: ./examples/refex/ex12.pp

Program Example12;

{ Program to demonstrate the Cos function. }

Var R : Real;

begin

 R:=Cos(Pi); { R:=-1 }

 R:=Cos(Pi/2); { R:=0 }

 R:=Cos(0); { R:=1 }

end.

75.12.78 CSeg

Synopsis: Return code segment.

Declaration: `function CSeg : Word`

Visibility: default

Description: CSeg returns the Code segment register. In Free Pascal, it returns always a zero, since Free Pascal is a 32/64 bit compiler.

Errors: None.

See also: DSeg ([1452](#)), Seg ([1536](#)), OfS ([1512](#)), Ptr ([1517](#))

Listing: ./examples/refex/ex13.pp

Program Example13;

{ Program to demonstrate the CSeg function. }

var W : word;

begin

 W:=CSeg; {W:=0, provided for compatibility,
 FPC is 32 bit.}

end.

75.12.79 Dec

Synopsis: Decrease value of variable.

Declaration: `procedure Dec(var X: TOrdinal)`
 `procedure Dec(var X: TOrdinal; Decrement: TOrdinal)`

Visibility: default

Description: Dec decreases the value of X with Decrement. If Decrement isn't specified, then 1 is taken as a default.

Dec can be used on typed pointers: in that case it decreases the value with Decrement the size of the type the pointer points to. This works independently of the setting of the \$POINTERMATH directive.

Errors: A range check can occur, or an underflow error, if an attempt is made to decrease `X` below its minimum value.

See also: `Inc` ([1484](#))

Listing: `./examples/refex/ex14.pp`

Program Example14;

{ Program to demonstrate the Dec function. }

Var

`I : Integer;`
`L : Longint;`
`W : Word;`
`B : Byte;`
`Si : ShortInt;`

begin

`I:=1;`
`L:=2;`
`W:=3;`
`B:=4;`
`Si:=5;`
`Dec (i); { i:=0 }`
`Dec (L,2); { L:=0 }`
`Dec (W,2); { W:=1 }`
`Dec (B,-2); { B:=6 }`
`Dec (Si,0); { Si:=5 }`

end.

75.12.80 Default

Synopsis: Return Default initialized value.

Declaration: `function Default(const T: AnyType) : AnyType`

Visibility: default

Description: `Default` is a compiler intrinsic: it returns for every type `T` a default value. In essence, this is a block of memory that is zeroed out. It can be used to correctly initialize most types (see below). Most importantly, it can be used to correctly initialize a managed type ([1335](#)). It also works using a generic type template.

This function must not be used on a type where 0 (or a block of zeroes) is not a valid value for that type. In particular, it must not be used on the file types or complex types that contain a file type. Similarly, if you use it on enumerated types and range types which do not have an element with ordinal value 0, the use of this function will result in invalid values. In both cases, the compiler will not give an error if you do use `Default` on variables of the mentioned types, but the result will be an invalid value.

See also: `TypeInfo` ([1563](#)), `Initialize` ([1489](#)), `Finalize` ([1467](#)), `ManagedTypes` ([1335](#))

75.12.81 DefaultAnsi2UnicodeMove

Synopsis: Standard widestring manager callback.

Declaration: `procedure DefaultAnsi2UnicodeMove(source: PChar; cp: TSystemCodePage;
var dest: unicodestring; len: SizeInt)`

Visibility: default

Description: `DefaultAnsi2UnicodeMove` is the standard callback used for the widestring manager when an ansistring must be converted to a unicodestring. It simply copies over all characters from the ansistring to the unicodestring, no conversion whatsoever is performed.

75.12.82 DefaultAnsi2WideMove

Synopsis: Standard implementation of Ansi to Widestring conversion routine.

Declaration: `procedure DefaultAnsi2WideMove(source: PChar; cp: TSystemCodePage;
var dest: widestring; len: SizeInt)`

Visibility: default

Description: `DefaultAnsi2WideMove` simply copies each character of the null-terminated ansi-string `Source` to the corresponding `WideChar` in `Dest`. At most `Len` characters will be copied.

Errors: None.

See also: `DefaultUnicode2AnsiMove` ([1449](#))

75.12.83 DefaultUnicode2AnsiMove

Synopsis: Standard widestring manager callback.

Declaration: `procedure DefaultUnicode2AnsiMove(source: PUnicodeChar;
var dest: RawByteString;
cp: TSystemCodePage; len: SizeInt)`

Visibility: default

Description: `DefaultUnicode2AnsiMove` is the standard callback used for the widestring manager when a Unicode string must be converted to an ansistring. It replaces all words with value < 256 with their value as ASCII code.

Errors: None.

See also: `WidestringManager` ([1418](#))

75.12.84 Delete

Synopsis: Delete elements (characters) from a string or dynamic array.

Declaration: `procedure Delete(var S: string; const Index: Integer;
const Count: Integer)
procedure Delete(var A: DynArrayType; const Index: Integer;
const Count: Integer)`

Visibility: default

Description: `Delete` removes `Count` characters from string `S`, starting at position `Index`. `Index` is 1-based. All characters after the deleted characters are shifted `Count` positions to the left, and the length of the string is adjusted.

For dynamic arrays, `Delete` removes `Count` elements from the array `A`, starting at position `Index`. `Index` is 0-based. All elements after the deleted elements are shifted `Count` positions to the left, and the length of the array is adjusted.

If the sum of `Index` and `Count` exceeds the length of the string or array, `Delete` removes the end of the string or array, starting at `Index`.

If `Index` is less than 1 or greater than the length of the string or array, or if `Count` is negative or zero, `Delete` does nothing.

See also: `Copy` ([1445](#)), `Pos` ([1515](#)), `Insert` ([1493](#))

Listing: `./examples/refex/ex15.pp`

Program `Example15`;

{ Program to demonstrate the Delete function. }

Var

`S : String;`

begin

`S := 'This is not easy !';`

`Delete (S, 9, 4); { S := 'This is easy !' }`

end.

75.12.85 Dispose

Synopsis: Free dynamically allocated memory.

Declaration: `procedure &Dispose(P: Pointer)`
`procedure &Dispose(P: TypedPointer; Des: TProcedure)`

Visibility: `default`

Description: The first form `Dispose` releases the memory allocated with a call to `New` ([1509](#)). The pointer `P` must be typed. The released memory is returned to the heap.

The second form of `Dispose` accepts as a first parameter a pointer to an object type, and as a second parameter the name of a destructor of this object. The destructor will be called, and the memory allocated for the object will be freed.

Errors: An runtime error will occur if the pointer doesn't point to a location in the heap.

See also: `New` ([1509](#)), `Getmem` ([1474](#)), `Freemem` ([1471](#))

Listing: `./examples/refex/ex16.pp`

Program `Example16`;

{ Program to demonstrate the Dispose and New functions. }

Type `SS = String[20];`

`AnObj = Object`

```

    I : integer;
    Constructor Init;
    Destructor Done;
    end;

Var
    P : ^SS;
    T : ^AnObj;

Constructor Anobj.Init;

begin
    Writeln ('Initializing an instance of AnObj !');
end;

Destructor AnObj.Done;

begin
    Writeln ('Destroying an instance of AnObj !');
end;

begin
    New (P);
    P^:='Hello, World !';
    Dispose (P);
    { P is undefined from here on !}
    New(T,Init);
    T^.i:=0;
    Dispose (T,Done);
end.

```

75.12.86 divide(variant,variant):variant

Synopsis: Implement division (/) operation on variants.

Declaration: `operator / (const op1: variant; const op2: variant) : variant`

Visibility: default

Description: The implementation of the division / operation is delegated to the variant manager with operation `opDivide`.

Errors: Execution of this operator may result in an exception if no variant manager is installed or if the types of the operand are not suitable for the operation.

See also: operator `*(variant, variant): variant` ([1335](#))

75.12.87 DoneCriticalSection

Synopsis: Clean up a critical section.

Declaration: `procedure DoneCriticalSection(var cs: TRTLCRITICALSECTION)`

Visibility: default

Description: `DoneCriticalSection` cleans up the critical section CS. After a call to `DoneCriticalSection`, the critical section can no longer be used with `EnterCriticalSection` ([1455](#)) or `LeaveCriticalSection` ([1499](#)), unless it is again initialized with `InitCriticalSection` ([1489](#))

See also: [InitCriticalSection \(1489\)](#), [EnterCriticalSection \(1455\)](#), [LeaveCriticalSection \(1499\)](#)

75.12.88 DoneThread

Synopsis: End the current thread.

Declaration: `procedure DoneThread`

Visibility: default

Description: `DoneThread` should be used to clean up the current thread. It performs the necessary housekeeping before the thread is actually ended. Under normal circumstances, you must not call this, it is called automatically.

See also: [BeginThread \(1430\)](#), [EndThread \(1455\)](#)

75.12.89 DSeg

Synopsis: Return data segment.

Declaration: `function DSeg : Word`

Visibility: default

Description: `DSeg` returns the data segment register. In Free Pascal, it returns always a zero, since Free Pascal is a 32/64 bit compiler.

Errors: None.

See also: [CSeg \(1447\)](#), [Seg \(1536\)](#), [Ofs \(1512\)](#), [Ptr \(1517\)](#)

Listing: `./examples/refex/ex17.pp`

Program Example17;

{ Program to demonstrate the DSeg function. }

Var

W : Word;

begin

W:=**DSeg**; *{W:=0, This function is provided for compatibility,
FPC is a 32 bit compiler.}*

end.

75.12.90 DumpExceptionBacktrace

Synopsis: Create backtrace.

Declaration: `procedure DumpExceptionBacktrace(var f: text)`

Visibility: default

Description: `DumpExceptionBackTrace` writes a backtrace of the current exception to the file `f`. If no exception is currently being raised, nothing is written. As much frames as available are written. If debug info is available, then file names and line numbers will be written as well.

Errors: No check is done to see whether `f` is opened for writing.

See also: [dump_stack \(1453\)](#)

75.12.91 Dump_Stack

Synopsis: Dump stack to the given text file.

Declaration: `procedure Dump_Stack(var f: text; fp: pointer; addr: CodePointer)`
`procedure Dump_Stack(var f: text; skipframes: LongInt)`

Visibility: default

Description: `Dump_Stack` prints a stack dump to the file `f`, with base frame pointer `bp`

Errors: The file `f` must be opened for writing or an error will occur.

See also: `get_caller_addr` (1479), `get_caller_frame` (1479), `get_frame` (1480)

75.12.92 DynArrayBounds

Synopsis: Return the bounds of the dynamic array.

Declaration: `function DynArrayBounds(a: Pointer; typeInfo: Pointer) : TBoundArray`

Visibility: default

Description: `DynArrayBounds` returns the bounds of all the dimensions of the dynamic array `a` with type information `typeInfo`.

The result is an array (zero-based) with the maximum valid index for each dimension in the array: the lower bound is not present in the result, it is always zero.

See also: `InitializeArray` (1492), `FinalizeArray` (1467), `CopyArray` (1446), `DynArraySize` (1454), `DynArrayClear` (1453), `DynArrayDim` (1453)

75.12.93 DynArrayClear

Synopsis: Clears a dynamic array.

Declaration: `procedure DynArrayClear(var a: Pointer; typeInfo: Pointer)`

Visibility: default

Description: `DynArrayClear` clears the array (`a`) using its type info (`typeInfo`). It is equal to setting the length to zero.

See also: `InitializeArray` (1492), `FinalizeArray` (1467), `CopyArray` (1446), `DynArraySize` (1454), `DynArrayDim` (1453), `DynArrayBounds` (1453)

75.12.94 DynArrayDim

Synopsis: Return the number of dimensions in a dynamic array.

Declaration: `function DynArrayDim(typeInfo: Pointer) : Integer`

Visibility: default

Description: `DynArrayDim` returns the number of dimensions in a dynamic array, using the type information (`typeInfo`) of the array.

See also: `InitializeArray` (1492), `FinalizeArray` (1467), `CopyArray` (1446), `DynArraySize` (1454), `DynArrayClear` (1453), `DynArrayBounds` (1453)

75.12.95 DynArrayIndex

Synopsis: Return pointer to indicated element.

Declaration: `function DynArrayIndex(a: Pointer; const indices: Array of SizeInt; typeInfo: Pointer) : Pointer`

Visibility: default

Description: `DynArrayIndex` returns a pointer to the element indicated by `indices` in dynamic array `a` with type information `typeInfo`. The length of `indices` must equal the number of dimensions of the array (as returned by `DynArrayDim` (1453)).

Errors: No bounds checking is performed, it is therefor possible to get an access violation if one of the indexes is out of range.

See also: `InitializeArray` (1492), `FinalizeArray` (1467), `CopyArray` (1446), `DynArraySize` (1454), `DynArrayClear` (1453), `DynArrayBounds` (1453), `DynArrayDim` (1453), `IsDynArrayRectangular` (1498)

75.12.96 DynArraySetLength

Synopsis: Set the length of a dynamic array.

Declaration: `procedure DynArraySetLength(var a: Pointer; typeInfo: Pointer; dimCnt: SizeInt; lengthVec: PSizeInt)`

Visibility: default

Description: `DynArraySetLength` sets the length of the dynamic array `a` to the first `dimCnt` lengths specified in the array `lengthVec`. The dynamic array type is described in `typeInfo` which points to a record of type `TDynArrayTypeInfo` (1387)

It should never be necessary to call this function directly, the standard `SetLength` (1538) function should be used instead.

Errors: If an invalid pointer is specified, an error may occur.

See also: `SetLength` (1538), `tdynarraytypeinfo` (1387)

75.12.97 DynArraySize

Synopsis: Return length of dynamic array.

Declaration: `function DynArraySize(a: pointer) : tdynarrayindex`

Visibility: default

Description: `DynArraySize` gets the number of elements in the array (`a`) the result is equal to `Length` (1500) for dynamic arrays.

See also: `InitializeArray` (1492), `FinalizeArray` (1467), `CopyArray` (1446), `DynArrayClear` (1453), `DynArrayDim` (1453), `DynArrayBounds` (1453)

75.12.98 EmptyMethod

Synopsis: Empty method alias.

Declaration: `procedure EmptyMethod`

Visibility: default

Description: `EmptyMethod` is meant for the compiler only. It should not be used directly.

75.12.99 EndThread

Synopsis: End the current thread.

Declaration: `procedure EndThread(ExitCode: DWord)`
`procedure EndThread`

Visibility: default

Description: `EndThread` ends the current thread. If `ExitCode` is supplied, it is returned as the exit code for the thread to a function waiting for the thread to terminate (`WaitForThreadTerminate` (1572)). If it is omitted, zero is used.

This function does not return.

See also: `WaitForThreadTerminate` (1572), `BeginThread` (1430)

75.12.100 EnterCriticalSection

Synopsis: Enter a critical section.

Declaration: `procedure EnterCriticalSection(var cs: TRTLCRITICALSECTION)`

Visibility: default

Description: `EnterCriticalSection` will suspend the current thread if another thread has currently entered the critical section. When the other thread has left the critical section (through `LeaveCriticalSection` (1499)), the current thread resumes execution. The result is that only 1 thread is executing code which is protected by a `EnterCriticalSection` and `LeaveCriticalSection` pair.

The critical section must have been initialized with `InitCriticalSection` (1489) prior to a call to `EnterCriticalSection`.

A call to `EnterCriticalSection` must always be matched by a call to `LeaveCriticalSection` (1499). To avoid problems, it is best to include the code to be execute in a `try...finally` block, as follows:

```
EnterCriticalSection(Section);
Try
    // Code to be protected goes here.
Finally
    LeaveCriticalSection(Section);
end;
```

For performance reasons it is best to limit the code between the entering and leaving of a critical section as short as possible.

See also: `InitCriticalSection` (1489), `DoneCriticalSection` (1451), `LeaveCriticalSection` (1499)

75.12.101 EnumResourceLanguages

Synopsis: Enumerate available languages for a resource of given type and name.

Declaration: `function EnumResourceLanguages(ModuleHandle: TFPResourceHMODULE;`
`ResourceType: PChar; ResourceName: PChar;`
`EnumFunc: EnumResLangProc; lParam: PtrInt)`
`: LongBool`

Visibility: default

Description: `EnumResourceLanguages` enumerates the available languages for a resource of given `ResourceName` and type `ResourceType` in the module `ModuleHandle`. For each language available, it calls `EnumFunc` and passes it `ModuleHandle`, the type of the resource `ResourceType`, the name of the resource `ResourceName`, the language ID, and `lParam`. It returns `False` if no resources are available for the specified resource type and module, or `True` if there are resources available.

Errors: None.

See also: `EnumResourceTypes` (1456), `EnumResourceNames` (1456), `EnumResourceLanguages` (1455)

75.12.102 EnumResourceNames

Synopsis: Enumerate available resource names for a specified resource type.

Declaration:

```
function EnumResourceNames (ModuleHandle: TFPResourceHMODULE;
                             ResourceType: PChar;
                             EnumFunc: EnumResNameProc; lParam: PtrInt)
                             : LongBool
```

Visibility: default

Description: `EnumResourceNames` enumerates the names of all resources of type `ResourceType` in the module `ModuleHandle`. For each resource available it calls `EnumFunc` and passes it `ModuleHandle`, the type of the resource `ResourceType`, the name of the resource, and `lParam`. It returns `False` if no resources are available for the specified resource type and module, or `True` if there are resources available.

Errors: None.

See also: `EnumResourceTypes` (1456), `EnumResourceLanguages` (1455)

75.12.103 EnumResourceTypes

Synopsis: Enumerate available resource types.

Declaration:

```
function EnumResourceTypes (ModuleHandle: TFPResourceHMODULE;
                             EnumFunc: EnumResTypeProc; lParam: PtrInt)
                             : LongBool
```

Visibility: default

Description: `EnumResourceTypes` enumerates the types of all resources in the module `ModuleHandle`. For each resource available it calls `EnumFunc` and passes it `ModuleHandle`, the type of the resource, and `lParam`. It returns `False` if no resources are available for the specified module, or `True` if there are resources available.

Errors: None.

See also: `EnumResourceNames` (1456), `EnumResourceLanguages` (1455)

75.12.104 EOF

Synopsis: Check for end of file.

Declaration:

```
function EOF (var f: File) : Boolean
function EOF (var t: Text) : Boolean
function EOF : Boolean
```

Visibility: default

Description: `Eof` returns `True` if the file-pointer has reached the end of the file, or if the file is empty. In all other cases `Eof` returns `False`. If no file `F` is specified, standard input is assumed.

Note that calling this function may cause your program to wait: to determine whether you are at EOF, it is necessary to read data. If the file descriptor is not a real file (for instance for standard input or sockets), then this call may seem to hang the program while it is waiting for data to appear or for the file descriptor to be closed.

Errors: Depending on the state of the `{ $I }` switch, a runtime error can be generated if there is an error. In the `{ $I- }` state, use `IOResult` to check for errors.

See also: `Eoln` ([1457](#)), `Assign` ([1423](#)), `Reset` ([1524](#)), `Rewrite` ([1525](#))

Listing: `./examples/refex/ex18.pp`

Program Example18;

{ Program to demonstrate the Eof function. }

Var T1,T2 : text;
C : Char;

begin
 { Set file to read from. Empty means from standard input. }
 assign (t1,paramstr(1));
 reset (t1);
 { Set file to write to. Empty means to standard output. }
 assign (t2,paramstr(2));
 rewrite (t2);
 While not eof(t1) **do**
 begin
 read (t1,C);
 write (t2,C);
 end;
 Close (t1);
 Close (t2);
end.

75.12.105 EOLn

Synopsis: Check for end of line.

Declaration: `function EOLn(var t: Text) : Boolean`
 `function EOLn : Boolean`

Visibility: default

Description: `EoLn` returns `True` if the file pointer has reached the end of a line, which is demarcated by a line-feed character (ASCII value 10), or if the end of the file is reached. In all other cases `EoLn` returns `False`. If no file `F` is specified, standard input is assumed. It can only be used on files of type `Text`.

Errors: None.

See also: `Eof` ([1456](#)), `Assign` ([1423](#)), `Reset` ([1524](#)), `Rewrite` ([1525](#))

Listing: `./examples/refex/ex19.pp`

Program Example19;

{ Program to demonstrate the Eoln function. }

begin

{ This program waits for keyboard input. }
{ It will print True when an empty line is put in,
and false when you type a non-empty line.
It will only stop when you press enter.}

While not Eoln do

Writeln (eoln);

end.

75.12.106 equal(variant,variant):Boolean

Synopsis: Implement = (equality) operation on variants.

Declaration: `operator =(const op1: variant; const op2: variant) : Boolean`

Visibility: default

Description: The implementation of the equality (=) operation is delegated to the variant manager with operation `opcmpeq`.

Errors: Execution of this operator may result in an exception if no variant manager is installed or if the types of the operand are not suitable for the operation.

See also: `operator <(variant, variant): boolean` ([1335](#))

75.12.107 Erase

Synopsis: Delete a file from disk.

Declaration: `procedure Erase(var f: File)`
`procedure Erase(var t: Text)`

Visibility: default

Description: `Erase` removes an unopened file from disk. The file should be assigned with `Assign`, but not opened with `Reset` or `Rewrite`

Errors: Depending on the state of the `{ $I }` switch, a runtime error can be generated if there is an error. In the `{ $I- }` state, use `IOResult` to check for errors.

See also: `Assign` ([1423](#))

Listing: `./examples/refex/ex20.pp`

Program Example20;

{ Program to demonstrate the Erase function. }

Var F : Text;

begin

{ Create a file with a line of text in it }
`Assign (F,'test.txt');`

```

Rewrite (F);
Writeln (F,'Try and find this when I'm finished !');
close (f);
{ Now remove the file }
Erase (f);
end.

```

75.12.108 Error

Synopsis: Generate run-time error.

Declaration: `procedure Error (RunTimeError: TRuntimeError)`

Visibility: default

Description: `Error` generates a run-time error with an exit code corresponding to `RunTimeError`. This function is implemented for Delphi compatibility, and is not used by the Free Pascal Run-Time Library.

See also: `RunError` ([1532](#)), `Halt` ([1481](#))

75.12.109 Exclude

Synopsis: Exclude element from a set if it is present.

Declaration: `procedure Exclude (var S: TSetType; E: TSetElement)`

Visibility: default

Description: `Exclude` removes `E` from the set `S` if it is included in the set. `E` should be of the same type as the base type of the set `S`.

Thus, the two following statements do the same thing:

```

S := S - [E];
Exclude (S, E);

```

Errors: If the type of the element `E` is not equal to the base type of the set `S`, the compiler will generate an error.

See also: `Include` ([1485](#))

Listing: `./examples/refex/ex111.pp`

```

program Example111;

```

```

{ Program to demonstrate the Include/Exclude functions }

```

Type

```

TEnumA = (aOne,aTwo,aThree);
TEnumAs = Set of TEnumA;

```

Var

```

SA : TEnumAs;

```

```

Procedure PrintSet(S : TEnumAs);

```

```

var

```

```

B : Boolean;

procedure DoEl(A : TEnumA; Desc : String);

begin
  If A in S then
    begin
      If B then
        Write(' ');
        B:=True;
        Write(Desc);
      end;
    end;

begin
  Write('[');
  B:=False;
  DoEl(aOne,'aOne');
  DoEl(aTwo,'aTwo');
  DoEl(aThree,'aThree');
  Writeln(']')
end;

begin
  SA:=[];
  Include(SA,aOne);
  PrintSet(SA);
  Include(SA,aThree);
  PrintSet(SA);
  Exclude(SA,aOne);
  PrintSet(SA);
  Exclude(SA,aTwo);
  PrintSet(SA);
  Exclude(SA,aThree);
  PrintSet(SA);
end.

```

75.12.110 Exit

Synopsis: Exit current subroutine.

Declaration: `procedure &Exit(const X: TAnyType)`
`procedure &Exit`

Visibility: default

Description: `Exit` exits the current subroutine, and returns control to the calling routine. If invoked in the main program routine, exit stops the program. The optional argument `X` allows to specify a return value, in the case `Exit` is invoked in a function. The function result will then be equal to `X`.

In Object Pascal or Delphi modes, if the `Exit` statement is surrounded by one or more `Try .. Finally` constructs, the `Finally` blocks are executed, which means that if the finally blocks are used to free resources, then these resources will also be freed when `Exit` is called.

Note that although `Exit` is a compiler intrinsic (i.e. is treated specially) it is defined as an identifier in the system unit, hence it can be redefined.

Errors: None.

See also: Halt ([1481](#))

Listing: ./examples/refex/ex21.pp

Program Example21;

{ Program to demonstrate the Exit function. }

Procedure DoAnExit (Yes : Boolean);

{ This procedure demonstrates the normal Exit }

begin

Writeln ('Hello from DoAnExit !');

if Yes **then**

begin

Writeln ('Bailing out early.');

exit;

end;

Writeln ('Continuing to the end.');

end;

Function Positive (Which : Integer) : Boolean;

*{ This function demonstrates the extra FPC feature of Exit :
You can specify a return value for the function }*

begin

if Which > 0 **then**

exit (True)

else

exit (False);

end;

begin

{ This call will go to the end }

 DoAnExit (False);

{ This call will bail out early }

 DoAnExit (True);

if Positive (-1) **then**

Writeln ('The compiler is nuts, -1 is not positive.')

else

Writeln ('The compiler is not so bad, -1 seems to be negative.');

end.

Listing: ./examples/refex/ex121.pp

{

Example 121:

Continue, break and exit are system procedures.

They can be redefined

}

procedure continue;

begin

Writeln ('Continue');

end;

Procedure Exit;

```
begin
  Writeln('exit');
end;
```

Procedure Break;

```
begin
  Writeln('Break');
end;
```

```
begin
  Repeat
    Continue;
  Break;
  exit;
  Until True;
end.
```

75.12.111 Exp

Synopsis: Exponentiate.

Declaration: `function Exp(d: ValReal) : ValReal`

Visibility: default

Description: `Exp` returns the exponent of X, i.e. the number e to the power X.

Errors: None.

See also: `Ln` ([1502](#))

Listing: `./examples/refex/ex22.pp`

Program Example22;

{ Program to demonstrate the Exp function. }

```
begin
  Writeln (Exp(1):8:2); { Should print 2.72 }
end.
```

75.12.112 Fail

Synopsis: Fail a constructor.

Declaration: `procedure Fail`

Visibility: default

Description: `Fail` can be used in a constructor for an object or class. It will exit the constructor at once, and the memory allocated for the constructor is freed. This mean that for objects allocated with `New` ([1509](#)), the resulting pointer is `Nil` and for classes, the object instance will be `Nil`.

See also: `TypeOf` ([1563](#)), `New` ([1509](#)), `Initialize` ([1489](#)), `Finalize` ([1467](#))

Listing: ./examples/refex/ex116.pp

```

program testfail;

{$mode objfpc}

Type
  TMyClass = Class
    Constructor Create;
  end;

Constructor TMyClass.Create;

begin
  Fail;
end;

var
  M : TMyClass;

begin
  M:=TMyClass.Create;
  WriteLn('M is nil : ',Not Assigned(M));
end.

```

75.12.113 FilePos

Synopsis: Get position in file.

Declaration: `function FilePos (var f: File) : Int64`

Visibility: default

Description: `FilePos` returns the current record position of the file-pointer in file `F`. It cannot be invoked with a file of type `Text`. A compiler error will be generated if this is attempted. Untyped files have a default record size of 128, if the second parameter to `Reset` ([1524](#)) isn't specified.

Errors: Depending on the state of the `{SI}` switch, a runtime error can be generated if there is an error. In the `{SI-}` state, use `IOResult` to check for errors.

See also: `Filesize` ([1464](#))

Listing: ./examples/refex/ex23.pp

```

Program Example23;

{ Program to demonstrate the FilePos function. }

Var F : File of Longint;
    L,FP : longint;

begin
  { Fill a file with data :
    Each position contains the position ! }
  Assign (F,'test.tmp');
  Rewrite (F);
  For L:=0 to 100 do

```

```

begin
  FP:=FilePos(F);
  Write (F,FP);
end;
Close (F);
Reset (F);
{ If all goes well, nothing is displayed here. }
While not (Eof(F)) do
begin
  FP:=FilePos (F);
  Read (F,L);
  if L<>FP then
    Writeln ('Something wrong: Got ',L,' on pos ',FP);
  end;
Close (F);
Erase (f);
end.

```

75.12.114 FileSize

Synopsis: Size of file.

Declaration: `function FileSize(var f: File) : Int64`

Visibility: default

Description: `Filesize` returns the total number of records in file `F`. It cannot be invoked with a file of type `Text`. (under Linux and Unix, this also means that it cannot be invoked on pipes). If `F` is empty, 0 is returned. Untyped files have a default record size of 128, if the second parameter to `Reset` ([1524](#)) isn't specified.

Note that the file must be open for this function to return a result.

Errors: Depending on the state of the `{SI}` switch, a runtime error can be generated if there is an error. In the `{SI-}` state, use `IOResult` to check for errors.

See also: `Filepos` ([1463](#))

Listing: `./examples/refex/ex24.pp`

Program Example24;

{ Program to demonstrate the FileSize function. }

```

Var F : File Of byte;
    L : File Of Longint;

begin
  Assign (F,paramstr(1));
  Reset (F);
  Writeln ('File size in bytes : ',FileSize(F));
  Close (F);
  Assign (L,paramstr (1));
  Reset (L);
  Writeln ('File size in Longints : ',FileSize(L));
  Close (f);
end.

```

75.12.115 FillByte

Synopsis: Fill memory region with 8-bit pattern.

Declaration: `procedure FillByte(var x; count: SizeInt; value: Byte)`

Visibility: default

Description: `FillByte` fills the memory starting at `X` with `Count` bytes with value equal to `Value`. This is useful for quickly zeroing out a memory location. When the size of the memory location to be filled out is a multiple of 2 bytes, it is better to use `Fillword` (1466), and if it is a multiple of 4 bytes it is better to use `FillDWord` (1466), these routines are optimized for their respective sizes.

Errors: No checking on the size of `X` is done.

See also: `Fillchar` (1465), `FillDWord` (1466), `Fillword` (1466), `Move` (1508)

Listing: `./examples/refex/ex103.pp`

Program Example103;

{ Program to demonstrate the FillByte function. }

```

Var S : String[10];
      I : Byte;

begin
  For i:=10 downto 0 do
    begin
      { Fill S with i bytes }
      FillByte (S,SizeOf(S),32);
      { Set Length }
      SetLength(S,I);
      Writeln (s,'*');
    end;
end.
```

75.12.116 FillChar

Synopsis: Fill memory region with certain character.

Declaration: `procedure FillChar(var x; count: SizeInt; Value: Byte)`
`procedure FillChar(var x; count: SizeInt; Value: Boolean)`
`procedure FillChar(var x; count: SizeInt; Value: Char)`

Visibility: default

Description: `Fillchar` fills the memory starting at `X` with `Count` bytes or characters with value equal to `Value`.

Errors: No checking on the size of `X` is done.

See also: `Fillword` (1466), `Move` (1508), `FillByte` (1465), `FillDWord` (1466)

Listing: `./examples/refex/ex25.pp`

Program Example25;

{ Program to demonstrate the FillChar function. }

```

Var S : String[10];
      I : Byte;
begin
  For i:=10 downto 0 do
    begin
      { Fill S with i spaces }
      FillChar (S,SizeOf(S),' ');
      { Set Length }
      SetLength(S,i);
      Writeln (s,'*');
    end;
end.

```

75.12.117 FillDWord

Synopsis: Fill memory region with 32-bit pattern.

Declaration: `procedure FillDWord(var x; count: SizeInt; value: DWord)`

Visibility: default

Description: Fillword fills the memory starting at X with Count DWords with value equal to Value. A DWord is 4 bytes in size.

Errors: No checking on the size of X is done.

See also: FillByte ([1465](#)), Fillchar ([1465](#)), Fillword ([1466](#)), Move ([1508](#))

Listing: ./examples/refex/ex104.pp

Program Example104;

{ Program to demonstrate the FillDWord function. }

```

Const
  ArraySize = 1000;

Var
  S : Array [1..ArraySize] of DWord;
  I : longint;

begin
  FillDWord(S,ArraySize,0);
  For I:=1 to ArraySize do
    if S[I]<>0 then
      Writeln('Position ',I,' not zeroed out');
end.

```

75.12.118 FillWord

Synopsis: Fill memory region with 16-bit pattern.

Declaration: `procedure FillWord(var x; count: SizeInt; Value: Word)`

Visibility: default

Description: `FillWord` fills the memory starting at `X` with `Count` words with value equal to `Value`. A word is 2 bytes in size.

Errors: No checking on the size of `X` is done.

See also: `FillChar` ([1465](#)), `Move` ([1508](#))

Listing: `./examples/refex/ex76.pp`

Program `Example76`;

{ Program to demonstrate the FillWord function. }

Var `W : Array[1..100] of Word`;

begin

{ Quick initialization of array W }

`FillWord(W,100,0);`

end.

75.12.119 Finalize

Synopsis: `Finalize` (clean up) memory block using RTTI.

Declaration: `procedure Finalize(var T: TAnyType; ACount: SizeInt)`

Visibility: default

Description: `Finalize` is a compiler intrinsic: it cleans up (finalizes) a memory area belonging to a variable `T` of a managed type ([1335](#)) (`TManagedType`). Finalizing means decreasing reference counts where necessary and generally zeroing out the memory area. It performs the opposite operation of `initialize` ([1489](#)).

The optional `ACount` parameter can be used to finalize an array of values.

For examples, see `Initialize` ([1489](#)).

See also: `Initialize` ([1489](#)), `Default` ([1448](#)), `TypeInfo` ([1563](#)), `ManagedTypes` ([1335](#))

75.12.120 FinalizeArray

Synopsis: `FinalizeArray` managed-type elements in array.

Declaration: `procedure FinalizeArray(p: Pointer; typeInfo: Pointer; count: SizeInt)`

Visibility: default

Description: `FinalizeArray` dereferences and clears managed types in the array pointed to by `p`. For this, it uses the type information of the elements as specified in `typeinfo`.

Under normal circumstances, this procedure should not be used, it is called automatically by the compiler when an array-typed variable containing managed types goes out of scope.

See also: `InitializeArray` ([1492](#)), `CopyArray` ([1446](#)), `DynArraySize` ([1454](#)), `DynArrayClear` ([1453](#)), `DynArrayDim` ([1453](#)), `DynArrayBounds` ([1453](#))

75.12.121 FindResource

Synopsis: Locate a resource and return a handle to it.

Declaration: `function FindResource(ModuleHandle: TFPResourceHMODULE;
ResourceName: PChar; ResourceType: PChar)
: TFPResourceHandle`
`function FindResource(ModuleHandle: TFPResourceHMODULE;
const ResourceName: AnsiString;
const ResourceType: AnsiString) : TFPResourceHandle`
`function FindResource(ModuleHandle: TFPResourceHMODULE;
const ResourceName: AnsiString;
ResourceType: PChar) : TFPResourceHandle`
`function FindResource(ModuleHandle: TFPResourceHMODULE;
ResourceName: PChar;
const ResourceType: AnsiString) : TFPResourceHandle`

Visibility: default

Description: `FindResource` searches for a resource with name `ResourceName` and of type `ResourceType` in the executable or library identified by `ModuleHandle`. It returns a `TResourceHandle` which can be used to load the resource with `LoadResource` (1503).

Errors: None. In case the resource was not found, 0 is returned.

See also: `FreeResource` (1472), `LoadResource` (1503), `SizeofResource` (1546), `LockResource` (1504), `UnlockResource` (1567), `FreeResource` (1472)

75.12.122 FindResourceEx

Synopsis: Find a resource based on type, name, language.

Declaration: `function FindResourceEx(ModuleHandle: TFPResourceHMODULE;
ResourceType: PChar; ResourceName: PChar;
Language: Word) : TFPResourceHandle`
`function FindResourceEx(ModuleHandle: TFPResourceHMODULE;
const ResourceType: AnsiString;
const ResourceName: AnsiString; Language: Word)
: TFPResourceHandle`
`function FindResourceEx(ModuleHandle: TFPResourceHMODULE;
ResourceType: PChar;
const ResourceName: AnsiString; Language: Word)
: TFPResourceHandle`
`function FindResourceEx(ModuleHandle: TFPResourceHMODULE;
const ResourceType: AnsiString;
ResourceName: PChar; Language: Word)
: TFPResourceHandle`

Visibility: default

Description: `FindResourceEx` looks in module `ModuleHandle` for a resource of type `ResourceType` and name `ResourceName` with language ID `Language`. Both `ResourceName` and `ResourceName` can be specified as a null-terminated array of characters, or as an `AnsiString`.

If the requested language/sublanguage is not found, then the search is conducted

1. with only primary language.

2.with the neutral language (LANG_NEUTRAL)

3.with the English language

If none of these has returned a match, then the first available language is returned.

If a match is found, a handle to the resource is returned. If none is found, an empty handle (nil or 0) is returned.

Errors: None.

75.12.123 float_raise

Synopsis: Raise floating point exception.

Declaration: `procedure float_raise(i: TFPUException)`
`procedure float_raise(i: TFPUExceptionMask)`

Visibility: default

Description: `float_raise` raises the floating point exceptions specified by `softfloat_exception_flags` (1417).

See also: `softfloat_exception_flags` (1417), `softfloat_exception_mask` (1417)

75.12.124 Flush

Synopsis: Write file buffers to disk.

Declaration: `procedure Flush(var t: Text)`

Visibility: default

Description: `Flush` empties the internal buffer of an opened file `F` and writes the contents to disk. The file is **not** closed as a result of this call.

Errors: Depending on the state of the `{ $I }` switch, a runtime error can be generated if there is an error. In the `{ $I- }` state, use `IOResult` to check for errors.

See also: `Close` (1437)

Listing: `./examples/refex/ex26.pp`

Program Example26;

{ Program to demonstrate the Flush function. }

Var F : Text;

begin

{ Assign F to standard output }

Assign (F,"");

Rewrite (F);

Writeln (F,'This line is written first, but appears later !');

*{ At this point the text is in the internal pascal buffer,
and not yet written to standard output }*

Writeln ('This line appears first, but is written later !');

*{ A writeln to 'output' always causes a flush – so this text is
written to screen }*

Flush (f);

```
{ At this point, the text written to F is written to screen. }
Write (F,'Finishing ');
Close (f); { Closing a file always causes a flush first }
Writeln ('off.');
```

end.

75.12.125 FlushThread

Synopsis: Flush all standard files.

Declaration: `procedure FlushThread`

Visibility: default

Description: `FlushThread` flushes any buffers from standard file descriptors such as standard input/output/error. It should normally not be called by user code, but is executed when a thread exits.

See also: `EndThread` ([1455](#))

75.12.126 FMADouble

Synopsis: Internal function, do not use.

Declaration: `function FMADouble(d1: Double; d2: Double; d3: Double) : Double`

Visibility: default

75.12.127 FMAExtended

Synopsis: Internal function, do not use.

Declaration: `function FMAExtended(e1: extended; e2: extended; e3: extended)
: extended`

Visibility: default

75.12.128 FMASingle

Synopsis: Internal function, do not use.

Declaration: `function FMASingle(s1: single; s2: single; s3: single) : single`

Visibility: default

75.12.129 FPower10

Synopsis: Fast multiply with a power of 10.

Declaration: `function FPower10(val: Extended; Power: LongInt) : Extended`

Visibility: default

Description: `FPower10` multiplies `val` with 10 to the power `Power`. It uses a fast algorithm to calculate the result.

75.12.130 Frac

Synopsis: Return fractional part of floating point value.

Declaration: `function Frac(d: ValReal) : ValReal`

Visibility: default

Description: `Frac` returns the non-integer part of `X`.

Errors: None.

See also: `Round` ([1529](#)), `Int` ([1493](#))

Listing: `./examples/refex/ex27.pp`

Program Example27;

{ Program to demonstrate the Frac function. }

Var R : Real;

begin

Writeln (**Frac** (123.456):0:3); *{ Prints 0.456 }*

Writeln (**Frac** (-123.456):0:3); *{ Prints -0.456 }*

end.

75.12.131 FreeLibrary

Synopsis: For compatibility with Delphi/Windows: Unload a library.

Declaration: `function FreeLibrary(Lib: TLibHandle) : Boolean`

Visibility: default

Description: `FreeLibrary` provides the same functionality as `UnloadLibrary` ([1566](#)), and is provided for compatibility with Delphi.

See also: `UnloadLibrary` ([1566](#))

75.12.132 Freemem

Synopsis: Release allocated memory.

Declaration: `procedure Freemem(p: pointer; Size: PtrUInt)`
 `function Freemem(p: pointer) : PtrUInt`

Visibility: default

Description: `Freemem` releases the memory occupied by the pointer `P`, of size `Count` (in bytes), and returns it to the heap. `P` should point to the memory allocated to a dynamic variable.

Errors: An error will occur when `P` doesn't point to the heap.

See also: `Getmem` ([1474](#)), `New` ([1509](#)), `Dispose` ([1450](#))

Listing: `./examples/refex/ex28.pp`

Program Example28;

{ Program to demonstrate the FreeMem and GetMem functions. }

Var P : Pointer;
MM : Longint;

begin
 { Get memory for P }
 GetMem (P,80);
 FillChar (P^,80,' ');
 FreeMem (P,80);
end.

75.12.133 Freememory

Synopsis: Alias for FreeMem ([1471](#)).

Declaration: `procedure Freememory(p: pointer; Size: PtrUInt)`
 `function Freememory(p: pointer) : PtrUInt`

Visibility: default

Description: FreeMemory is an alias for FreeMem ([1471](#)).

See also: FreeMem ([1471](#))

75.12.134 FreeResource

Synopsis: Free a loaded resource.

Declaration: `function FreeResource(ResData: TFPResourceHGLOBAL) : LongBool`

Visibility: default

Description: FreeResource unloads the resource identified by ResData from memory. The resource must have been loaded by LoadResource ([1503](#)). It returns True if the operation was successful, False otherwise.

Errors: On error, False is returned.

See also: FindResource ([1468](#)), LoadResource ([1503](#)), SizeofResource ([1546](#)), LockResource ([1504](#)), UnlockResource ([1567](#)), FreeResource ([1472](#))

75.12.135 Get8087CW

Declaration: `function Get8087CW : Word`

Visibility: default

75.12.136 GetCPUCount

Synopsis: Return the number of cores on the system.

Declaration: `function GetCPUCount : LongWord`

Visibility: default

Description: `GetCPUCount` returns the number of CPU cores on the system. Whether these are physically separate CPUs or cores on a single CPU is deliberately undefined.

See also: `CPUCount` ([1335](#))

75.12.137 `GetCurrentThreadId`

Synopsis: Return the id of the currently running thread.

Declaration: `function GetCurrentThreadId : TThreadId`

Visibility: default

Description: `GetCurrentThreadId` returns the ID of the currently running thread. It can be used in calls such as `KillThread` ([1499](#)) or `ThreadSetPriority` ([1560](#))

Errors: None.

See also: `KillThread` ([1499](#)), `ThreadSetPriority` ([1560](#))

75.12.138 `GetDir`

Synopsis: Return the current directory.

Declaration: `procedure GetDir(drivenr: Byte; var dir: shortstring); Overload`
`procedure GetDir(drivenr: Byte; var dir: RawByteString); Overload`
`procedure GetDir(drivenr: Byte; var dir: unicodestring); Overload`

Visibility: default

Description: `GetDir` returns in `dir` the current directory on the drive `drivenr`, where {`drivenr`} is 1 for the first floppy drive, 3 for the first hard disk etc. A value of 0 returns the directory on the current disk. On Linux and Unix systems, `drivenr` is ignored, as there is only one directory tree.

Errors: An error is returned under dos, if the drive requested isn't ready.

See also: `Chdir` ([1436](#))

Listing: `./examples/refex/ex29.pp`

Program Example29;

{ Program to demonstrate the GetDir function. }

Var S : **String**;

begin

GetDir (0,S);

Writeln ('Current directory is : ',S);

end.

75.12.139 GetDynLibsManager

Synopsis: Return currently active dynamic library support handler.

Declaration: `procedure GetDynLibsManager (var Manager: TDynLibsManager)`

Visibility: default

Description: `GetDynLibsManager` returns the currently active dynamic library support handler. This handler has normally been set by inclusion of the `#rtl.dynlibs` (754) unit.

See also: `#rtl.dynlibs` (754), `SetDynLibsManager` (1537)

75.12.140 GetFPCHeapStatus

Synopsis: Return FPC heap manager status information.

Declaration: `function GetFPCHeapStatus : TFPCHeapStatus`

Visibility: default

Description: Return FPC heap manager status information.

75.12.141 GetHeapStatus

Synopsis: Return the memory manager heap status.

Declaration: `function GetHeapStatus : THeapStatus`

Visibility: default

75.12.142 GetLoadErrorStr

Synopsis: Return an error describing the last library loading error.

Declaration: `function GetLoadErrorStr : string`

Visibility: default

Description: `GetLoadErrorStr` returns an error string describing the last library loading error. This function must be called before any other OS calls are performed.

Errors: None.

See also: `LoadLibrary` (1503), `SafeLoadLibrary` (1533)

75.12.143 GetMem

Synopsis: Allocate new memory on the heap.

Declaration: `procedure Getmem(out p: pointer; Size: PtrUInt)`
`function GetMem(size: PtrUInt) : pointer`

Visibility: default

Description: `Getmem` reserves `Size` bytes memory on the heap, and returns a pointer to this memory in `p`. What happens if no more memory is available, depends on the value of the variable `ReturnNilIfGrowHeapFails` (1417): if the variable is `True` then `Nil` is returned. If the variable is `False`, a run-time error is generated. The default value is `False`, so by default an error is generated.

The newly allocated memory is not initialized in any way, and may contain garbage data. It must be cleared with a call to `FillChar` (1465) or `FillWord` (1466).

For an example, see `Freemem` (1471).

Errors: None.

See also: `Freemem` (1471), `Dispose` (1450), `New` (1509), `returnnilifgrowheapfails` (1417), `MemSize` (1507)

75.12.144 GetMemory

Synopsis: Alias for `GetMem` (1474).

Declaration: `procedure Getmemory(out p: pointer; Size: PtrUInt)`
`function GetMemory(size: PtrUInt) : pointer`

Visibility: default

Description: `Getmemory` is an alias for `GetMem` (1474).

See also: `GetMem` (1474)

75.12.145 GetMemoryManager

Synopsis: Return current memory manager.

Declaration: `procedure GetMemoryManager(var MemMgr: TMemoryManager)`

Visibility: default

Description: `GetMemoryManager` stores the current Memory Manager record in `MemMgr`.

For an example, see the programmer's guide.

Errors: None.

See also: `SetMemoryManager` (1539), `IsMemoryManagerSet` (1499)

75.12.146 GetMXCSR

Declaration: `function GetMXCSR : DWord`

Visibility: default

75.12.147 GetProcAddress

Synopsis: For compatibility with Delphi/Windows: Get the address of a procedure.

Declaration: `function GetProcAddress(Lib: TLibHandle; const ProcName: AnsiString)`
`: Pointer`

Visibility: default

Description: `GetProcAddress` provides the same functionality as `GetProcedureAddress` (1476), and is provided for compatibility with Delphi.

75.12.148 GetProcAddress

Synopsis: Get the address of a procedure or symbol in a dynamic library.

Declaration: `function GetProcAddress(Lib: TLibHandle;
 const ProcName: AnsiString) : Pointer
 function GetProcAddress(Lib: TLibHandle; Ordinal: TOrdinalEntry)
 : Pointer`

Visibility: default

Description: `GetProcAddress` returns a pointer to the location in memory of the symbol `ProcName` or ordinal value `Ordinal` in the dynamically loaded library specified by its handle `lib`. If the symbol cannot be found or the handle is invalid, `Nil` is returned.

On Windows, only an exported procedure or function can be searched this way. On Unix platforms the location of any exported symbol can be retrieved this way.

Only windows and OS/2 support getting the address of a function using an ordinal value.

Errors: If the symbol cannot be found, `Nil` is returned.

See also: `LoadLibrary` ([1503](#)), `UnLoadLibrary` ([1566](#))

75.12.149 GetProcessID

Synopsis: Get the current process ID.

Declaration: `function GetProcessID : SizeUInt`

Visibility: default

Description: `GetProcessID` returns the current process ID. The meaning of the return value of this call is system dependent.

Errors: None.

See also: `GetThreadID` ([1477](#))

75.12.150 GetResourceManager

Synopsis: Return the currently active resource manager.

Declaration: `procedure GetResourceManager(var Manager: TResourceManager)`

Visibility: default

Description: `GetResourceManager` returns the currently active resource manager record in `Manager`. There is always an active resource manager record.

Errors: None.

See also: `TResourceManager` ([1396](#)), `SetResourceManager` ([1541](#))

75.12.151 GetSSECSR

Declaration: `function GetSSECSR : DWord`

Visibility: default

75.12.152 GetTextAutoFlush

Declaration: `function GetTextAutoFlush(var T: Text) : Boolean`

Visibility: default

75.12.153 GetTextCodePage

Synopsis: Get the codepage used in a text file.

Declaration: `function GetTextCodePage(var T: Text) : TSystemCodePage`

Visibility: default

Description: `GetTextCodePage` returns the codepage that the text file `T` uses. All strings written to the file will be converted to the indicated codepage. By default, the codepage is set to `CP_ACP`.

Errors: None.

See also: `TextRec` ([1389](#)), `SetTextCodePage` ([1543](#))

75.12.154 GetThreadID

Synopsis: Get the current Thread ID.

Declaration: `function GetThreadID : TThreadID`

Visibility: default

Description: `GetThreadID` returns the current process ID. The meaning of the return value of this call is system dependent.

See also: `GetProcessID` ([1476](#))

75.12.155 GetThreadManager

Synopsis: Return the current thread manager.

Declaration: `function GetThreadManager(var TM: TThreadManager) : Boolean`

Visibility: default

Description: `GetThreadManager` returns the currently used thread manager in `TM`.

For more information about thread programming, see the programmer's guide.

See also: `SetThreadManager` ([1544](#)), `TThreadManager` ([1400](#))

75.12.156 GetTypeKind

Synopsis: Return type kind for a type.

Declaration: `function GetTypeKind(const T: AnyType) : TTypeKind`

Visibility: default

Description: `GetTypeKind` is a compiler intrinsic: it returns the type kind for the type `T`. In difference with the `TypeInfo` ([1563](#)), if no type information was yet generated for the type, this statement will not ensure that the type information is available: the compiler knows the correct value, and will directly insert it into the code as a constant.

See also: `Default` ([1448](#)), `TypeInfo` ([1563](#)), `TypeOf` ([1563](#)), `Initialize` ([1489](#)), `Finalize` ([1467](#))

75.12.157 GetUnicodeStringManager

Synopsis: Return a copy of the currently active UnicodeString manager.

Declaration: `procedure GetUnicodeStringManager(var Manager: TUnicodeStringManager)`

Visibility: default

Description: `GetUnicodeStringManager` returns a copy of the currently active Unicode string manager in `Manager`.

UnicodeStrings are implemented in different ways on different platforms. Therefore, the Free Pascal Runtime library has no fixed implementation of widestring routines. Instead, it defines a `UnicodeStringManager` record, with callbacks that can be set to an implementation which is most efficient on the current platform. On windows, standard Windows routines will be used. On Unix and Linux, an implementation based on the C library is available (in unit `cwstring`).

It is possible to implement a custom unicodestring manager, optimized for the current application, without having to recompile the complete Run-Time Library.

See also: `SetUnicodeStringManager` ([1544](#)), `TUnicodeStringManager` ([1403](#))

75.12.158 GetVariantManager

Synopsis: Return the current variant manager.

Declaration: `procedure GetVariantManager(var VarMgr: tvariantmanager)`

Visibility: default

Description: `GetVariantManager` returns the current variant manager in `varmgr`.

See also: `SetVariantManager` ([1544](#))

75.12.159 GetWideStringManager

Synopsis: Return a copy of the currently active widestring manager.

Declaration: `procedure GetWideStringManager(var Manager: TUnicodeStringManager)`

Visibility: default

Description: `GetWideStringManager` returns a copy of the currently active widestring manager in `Manager`.

WideStrings are implemented in different ways on different platforms. Therefore, the Free Pascal Runtime library has no fixed implementation of widestring routines. Instead, it defines a `WideString` manager record, with callbacks that can be set to an implementation which is most efficient on the current platform. On windows, standard Windows routines will be used. On Unix and Linux, an implementation based on the C library is available (in unit `cwstring`).

It is possible to implement a custom widestring manager, optimized for the current application, without having to recompile the complete Run-Time Library.

See also: `SetWideStringManager` ([1545](#)), `TWideStringManager` ([1412](#))

75.12.160 `get_caller_addr`

Synopsis: Return the address of the caller.

Declaration: `function get_caller_addr(framebp: pointer; addr: CodePointer)
: CodePointer`

Visibility: default

Description: `get_caller_frame` returns a pointer to address (the return address) of the caller of the routine which has as frame `framebp`.

See also: `get_frame` ([1480](#)), `get_caller_frame` ([1479](#)), `Dump_Stack` ([1453](#))

75.12.161 `get_caller_frame`

Synopsis: Return the frame pointer of the caller.

Declaration: `function get_caller_frame(framebp: pointer; addr: CodePointer) : pointer`

Visibility: default

Description: `get_caller_frame` returns a pointer to the frame of the caller of the routine which has as frame `framebp`.

See also: `get_caller_addr` ([1479](#)), `get_frame` ([1480](#)), `Dump_Stack` ([1453](#))

75.12.162 `get_caller_stackinfo`

Synopsis: Return caller stack infomation.

Declaration: `procedure get_caller_stackinfo(var framebp: pointer;
var addr: CodePointer)`

Visibility: default

Description: `get_caller_stackinfo` returns caller address in `addr` and frame base pointer in `framebp`.

See also: `CaptureBacktrace` ([1436](#)), `Get_pc_addr` ([1480](#)), `get_caller_addr` ([1479](#)), `get_caller_frame` ([1479](#))

75.12.163 `get_cmdline`

Synopsis: Return the command-line as a null-terminated string.

Declaration: `function get_cmdline : PChar`

Visibility: default

Description: `get_cmdline` returns the complete command-line as a null-terminated string. It is not recommended to use this function, since it builds a complete value from the actual command-line arguments. Instead, `ParamCount` ([1513](#)) and `ParamStr` ([1514](#)) should be used.

See also: `ParamCount` ([1513](#)), `ParamStr` ([1514](#))

75.12.164 get_frame

Synopsis: Return the current frame.

Declaration: `function get_frame : pointer`

Visibility: default

Description: `get_frame` returns a pointer to the current stack frame.

See also: `get_caller_addr` ([1479](#)), `get_caller_frame` ([1479](#))

75.12.165 Get_pc_addr

Synopsis: Get Program Counter address.

Declaration: `function Get_pc_addr : CodePointer`

Visibility: default

Description: `Get_pc_addr` returns the program counter address (current execution address).

See also: `CaptureBacktrace` ([1436](#)), `get_caller_stackinfo` ([1479](#)), `get_caller_addr` ([1479](#)), `get_caller_frame` ([1479](#))

75.12.166 greaterthan(variant,variant):Boolean

Synopsis: Implement > (greater than) operation on variants.

Declaration: `operator >(const op1: variant; const op2: variant) : Boolean`

Visibility: default

Description: The implementation of the "greater than" comparison (>) operation is delegated to the variant manager with operation `opcmpgt`.

Errors: Execution of this operator may result in an exception if no variant manager is installed or if the types of the operand are not suitable for the operation.

See also: `operator <(variant, variant): boolean` ([1335](#))

75.12.167 greaterthanorequal(variant,variant):Boolean

Synopsis: Implement >= (greater than or equal) operation on variants.

Declaration: `operator >=(const op1: variant; const op2: variant) : Boolean`

Visibility: default

Description: The implementation of the "greater than or equal" comparison (>=) operation is delegated to the variant manager with operation `opcmpge`.

Errors: Execution of this operator may result in an exception if no variant manager is installed or if the types of the operand are not suitable for the operation.

See also: `operator <(variant, variant): boolean` ([1335](#))

75.12.168 Halt

Synopsis: Stop program execution.

Declaration: `procedure Halt (errnum: LongInt)`
`procedure Halt`

Visibility: default

Description: `Halt` stops program execution and returns control to the calling program. The optional argument `Errnum` specifies an exit value. If omitted, zero is returned.

Note that this skips any try/finally (implicit or explicit) or try/except blocks, thus may result in memory leaks. Finalization sections of units will be executed.

Errors: None.

See also: `Exit` ([1460](#))

Listing: `./examples/refex/ex30.pp`

Program Example30;

{ Program to demonstrate the Halt function. }

```
begin
  Writeln ('Before Halt.');
```

Halt (1); *{ Stop with exit code 1 }*

```
  Writeln ('After Halt doesn"t get executed.');
```

end.

75.12.169 HexStr

Synopsis: Convert integer value to string with hexadecimal representation.

Declaration: `function HexStr (Val: LongInt; cnt: Byte) : shortstring`
`function HexStr (Val: Int64; cnt: Byte) : shortstring`
`function HexStr (Val: QWord; cnt: Byte) : shortstring`
`function HexStr (Val: Pointer) : shortstring`

Visibility: default

Description: `HexStr` returns a string with the hexadecimal representation of `Value`. The string has exactly `cnt` characters. (i.e. only the `cnt` rightmost nibbles are taken into account) To have a complete representation of a Longint-type value, 8 nibbles are needed, i.e. `cnt=8`.

Errors: None.

See also: `Str` ([1549](#)), `Val` ([1570](#)), `BinStr` ([1430](#))

Listing: `./examples/refex/ex81.pp`

Program example81;

{ Program to demonstrate the HexStr function }

Const Value = 45678;

Var I : longint;

```

begin
  For I:=1 to 10 do
    Writeln (HexStr(Value,I));
end.

```

75.12.170 Hi

Synopsis: Return high byte/word/nibble of value.

Declaration:

```

function Hi(b: Byte) : Byte
function Hi(i: Integer) : Byte
function Hi(w: Word) : Byte
function Hi(l: LongInt) : Word
function Hi(l: DWord) : Word
function Hi(i: Int64) : DWord
function Hi(q: QWord) : DWord

```

Visibility: default

Description: `Hi` returns the high nibble, byte or word or longword from `X`, depending on the size of `X`.

Table 75.21:

Size	Return value
8	Byte, High nibble
16	Byte, High byte
32	Word, High word
64	Cardinal, High DWord

Note that in Delphi or TP, this function always treats its argument as if it was a `Word`, so the results may differ from FPC.

Errors: None.

See also: [Lo \(1502\)](#)

Listing: ./examples/refex/ex31.pp

Program Example31;

{ Program to demonstrate the Hi function. }

var

```

L : Longint;
W : Word;
B : Byte;

```

begin

```

L:=1 Shl 16; { = $10000 }
W:=1 Shl 8; { = $100 }
B:=1 Shl 4; { = $10 }
Writeln (Hi(L)); { Prints 1 }
Writeln (Hi(W)); { Prints 1 }

```

```

Writeln (Hi(B)); { Prints 1 }
end.

```

75.12.171 High

Synopsis: Return highest index of open array or enumerated.

Declaration: `function High(Arg: TypeOrVariable) : TOrdinal`

Visibility: default

Description: The return value of `High` depends on it's argument:

- 1.If the argument is an ordinal type, `High` returns the highest value in the range of the given ordinal type.
- 2.If the argument is an array type or an array type variable then `High` returns the highest possible value of it's index. For dynamic arrays, it returns the same as `Length -1`, meaning that it reports -1 for empty arrays.
- 3.If the argument is an open array identifier in a function or procedure, then `High` returns the highest index of the array, as if the array has a zero-based index. If the array is empty, then -1 is returned.
- 4.If the argument is a set type then it returns the highest value of the underlying ordinal type.

The return type is always the same type as the type of the argument (This can lead to some nasty surprises !).

Errors: None.

See also: [Low \(1506\)](#), [Ord \(1512\)](#), [Pred \(1517\)](#), [Succ \(1554\)](#)

Listing: `./examples/refex/ex80.pp`

Program example80;

{ Example to demonstrate the High and Low functions. }

```

Type TEnum = ( North, East, South, West );
      TRange = 14..55;
      TArray = Array [2..10] of Longint;

```

Function Average (Row : **Array of** Longint) : Real;

```

Var I : longint;
     Temp : Real;

```

```

begin
  Temp := Row[0];
  For I := 1 to High(Row) do
    Temp := Temp + Row[i];
  Average := Temp / (High(Row)+1);
end;

```

```

Var A : TEnum;
     B : TRange;
     C : TArray;

```

```

    I : longint;

begin
    Writeln ('TEnum goes from : ', Ord(Low(TEnum)), ' to ', Ord(high(TEnum)), '.');
    Writeln ('A goes from : ', Ord(Low(A)), ' to ', Ord(high(A)), '.');
    Writeln ('TRange goes from : ', Ord(Low(TRange)), ' to ', Ord(high(TRange)), '.');
    Writeln ('B goes from : ', Ord(Low(B)), ' to ', Ord(high(B)), '.');
    Writeln ('TArray index goes from : ', Ord(Low(TArray)), ' to ', Ord(high(TArray)), '.');
    Writeln ('C index goes from : ', Low(C), ' to ', high(C), '.');
    For I:=Low(C) to High(C) do
        C[i]:=I;
    Writeln ('Average : ', Average(c));
    Write ('Type of return value is always same as type of argument:');
    Writeln(high(high(word)));
end.

```

75.12.172 HINSTANCE

Synopsis: Windows compatibility type for use in resources.

Declaration: `function HINSTANCE : TFPResourceHMODULE`

Visibility: default

Description: This is an opaque type.

75.12.173 Inc

Synopsis: Increase value of integer variable.

Declaration: `procedure Inc(var X: TOrdinal)`
`procedure Inc(var X: TOrdinal; Increment: TOrdinal)`

Visibility: default

Description: `Inc` increases the value of `X` with `Increment`. If `Increment` isn't specified, then 1 is taken as a default.

`Inc` can be used on typed pointers: in that case it increases the value with `Increment` the size of the type the pointer points to. This works independently of the setting of the `$POINTERMATH` directive.

Errors: If range checking is on, then A range check can occur, or an overflow error, when an attempt is made to increase `X` over its maximum value.

See also: `Dec` ([1447](#))

Listing: `./examples/refex/ex32.pp`

Program Example32;

{ Program to demonstrate the Inc function. }

Const

```

C : Cardinal = 1;
L : Longint = 1;
I : Integer = 1;
W : Word = 1;

```

```

B : Byte = 1;
SI : ShortInt = 1;
CH : Char = 'A';

begin
  Inc (C); { C:=2 }
  Inc (L,5); { L:=6 }
  Inc (I,-3); { I:=-2 }
  Inc (W,3); { W:=4 }
  Inc (B,100); { B:=101 }
  Inc (SI,-3); { SI:=-2 }
  Inc (CH,1); { ch:='B' }
end.

```

75.12.174 Include

Synopsis: Include element in set if it was not yet present.

Declaration: `procedure Include (var S: TSetType; E: TSetElement)`

Visibility: default

Description: `Include` includes `E` in the set `S` if it is not yet part of the set. `E` should be of the same type as the base type of the set `S`.

Thus, the two following statements do the same thing:

```

S := S + [E];
Include (S, E);

```

For an example, see `Exclude` ([1459](#))

Errors: If the type of the element `E` is not equal to the base type of the set `S`, the compiler will generate an error.

See also: `Exclude` ([1459](#))

75.12.175 IndexByte

Synopsis: Search for a byte in a memory range.

Declaration: `function IndexByte (const buf; len: SizeInt; b: Byte) : SizeInt`

Visibility: default

Description: `IndexByte` searches the memory at `buf` for maximally `len` positions for the byte `b` and returns it's position if it found one. If `b` is not found then `-1` is returned. The position is zero-based.

Errors: `Buf` and `Len` are not checked to see if they are valid values.

See also: `IndexChar` ([1486](#)), `IndexDWord` ([1487](#)), `IndexWord` ([1488](#)), `CompareByte` ([1438](#))

Listing: `./examples/refex/ex105.pp`

Program Example105;

{ Program to demonstrate the IndexByte function. }

Const

 ArraySize = 256;
 MaxValue = 256;

Var

 Buffer : **Array**[1..ArraySize] **of** Byte;
 I,J : longint;
 K : Byte;

begin

Randomize;

For I:=1 **To** ArraySize **do**

 Buffer[I]:=Random(MaxValue);

For I:=1 **to** 10 **do**

begin

 K:=Random(MaxValue);

 J:=IndexByte(Buffer,ArraySize,K);

if J=-1 **then**

 Writeln('Value ',K,' was not found in buffer.')

else

 Writeln('Found ',K,' at position ',J,' in buffer');

end;

end.

75.12.176 IndexChar

Synopsis: Search for a character in a memory range.

Declaration: `function IndexChar(const buf; len: SizeInt; b: Char) : SizeInt`

Visibility: default

Description: `IndexChar` searches the memory at `buf` for maximally `len` positions for the character `b` and returns it's position if it found one. If `b` is not found then -1 is returned. The position is zero-based. The `IndexChar0` variant stops looking if a null character is found, and returns -1 in that case.

Errors: `Buf` and `Len` are not checked to see if they are valid values.

See also: `IndexByte` ([1485](#)), `IndexDWord` ([1487](#)), `IndexWord` ([1488](#)), `CompareChar` ([1439](#))

Listing: `./examples/refex/ex108.pp`

Program Example108;

{ Program to demonstrate the IndexChar function. }

Const

 ArraySize = 1000;
 MaxValue = 26;

Var

 Buffer : **Array**[1..ArraySize] **of** Char;
 I,J : longint;

```

K : Char;

begin
  Randomize;
  For I:=1 To ArraySize do
    Buffer[I]:=chr(Ord('A')+Random(MaxValue));
  For I:=1 to 10 do
    begin
      K:=chr(Ord('A')+Random(MaxValue));
      J:=IndexChar(Buffer,ArraySize,K);
      if J=-1 then
        Writeln('Value ',K,' was not found in buffer.')
      else
        Writeln('Found ',K,' at position ',J,' in buffer');
      end;
    end.

```

75.12.177 IndexChar0

Synopsis: Return index of a character in null-terminated array of char.

Declaration: `function IndexChar0(const buf; len: SizeInt; b: Char) : SizeInt`

Visibility: default

Description: `IndexChar0` returns the index of the character `b` in the null-terminated array `Buf`. At most `len` characters will be searched, or the null character if it is encountered first. If the character is not found, -1 is returned.

Errors: On error, -1 is returned.

See also: `IndexByte` ([1485](#)), `IndexChar` ([1486](#)), `IndexWord` ([1488](#)), `IndexDWord` ([1487](#)), `CompareChar0` ([1441](#))

75.12.178 IndexDWord

Synopsis: Search for a DWord value in a memory range.

Declaration: `function IndexDWord(const buf; len: SizeInt; b: DWord) : SizeInt`

Visibility: default

Description: `IndexDWord` searches the memory at `buf` for maximally `len` positions for the DWord `DW` and returns it's position if it found one. If `DW` is not found then -1 is returned. The position is zero-based.

Errors: `Buf` and `Len` are not checked to see if they are valid values.

See also: `IndexByte` ([1485](#)), `IndexChar` ([1486](#)), `IndexWord` ([1488](#)), `CompareDWord` ([1441](#))

Listing: `./examples/refex/ex106.pp`

Program Example106;

{ Program to demonstrate the IndexDWord function. }

Const

```

  ArraySize = 1000;
  MaxValue = 1000;

```



```

Var
  Buffer : Array[1..ArraySize] of DWord;
  I,J : longint;
  K : DWord;

begin
  Randomize;
  For I:=1 To ArraySize do
    Buffer[I]:=Random(MaxValue);
  For I:=1 to 10 do
    begin
      K:=Random(MaxValue);
      J:=IndexDWord(Buffer,ArraySize,K);
      if J=-1 then
        Writeln('Value ',K,' was not found in buffer.')
      else
        Writeln('Found ',K,' at position ',J,' in buffer');
      end;
    end.

```

75.12.179 IndexQWord

Synopsis: Return the position of a QWord in a memory range.

Declaration: `function IndexQWord(const buf; len: SizeInt; b: QWord) : SizeInt`

Visibility: default

Description: `IndexQWord` checks the first `len` qwords starting at `Buf`, and returns the position (zero-based) of `b`. If `b` does not appear in the first `len` qwords, then -1 is returned.

Note that the search is done on QWord boundaries, but that the address of `buf` need not be on a QWord boundary.

Errors: No check is done to see whether the indicated memory range is valid. If it is not, a run-error or exception may be triggered.

See also: `IndexDWord` ([1487](#))

75.12.180 Indexword

Synopsis: Search for a WORD value in a memory range.

Declaration: `function Indexword(const buf; len: SizeInt; b: Word) : SizeInt`

Visibility: default

Description: `IndexWord` searches the memory at `buf` for maximally `len` positions for the Word `W` and returns it's position if it found one. If `W` is not found then -1 is returned.

Errors: `Buf` and `Len` are not checked to see if they are valid values.

See also: `IndexByte` ([1485](#)), `IndexDWord` ([1487](#)), `IndexChar` ([1486](#)), `CompareWord` ([1442](#))

Listing: `./examples/refex/ex107.pp`

Program Example107;

{ Program to demonstrate the IndexWord function. }

Const

ArraySize = 1000;
MaxValue = 1000;

Var

Buffer : **Array**[1..ArraySize] **of** Word;
I,J : longint;
K : Word;

begin

Randomize;

For I:=1 **To** ArraySize **do**

Buffer[I]:=Random(MaxValue);

For I:=1 **to** 10 **do**

begin

K:=Random(MaxValue);

J:=IndexWord(Buffer,ArraySize,K);

if J=-1 **then**

Writeln('Value ',K,' was not found in buffer.')

else

Writeln('Found ',K,' at position ',J,' in buffer');

end;

end.

75.12.181 InitCriticalSection

Synopsis: Initialize a critical section.

Declaration: `procedure InitCriticalSection(var cs: TRTLCRITICALSECTION)`

Visibility: default

Description: `InitCriticalSection` initializes a critical section CS for use. Before using a critical section with `EnterCriticalSection` (1455) or `LeaveCriticalSection` (1499) the critical section should be initialized with `InitCriticalSection`.

When a critical section is no longer used, it should be disposed of with `DoneCriticalSection` (1451)

See also: `DoneCriticalSection` (1451), `EnterCriticalSection` (1455), `LeaveCriticalSection` (1499)

75.12.182 Initialize

Synopsis: Initialize memory block using RTTI.

Declaration: `procedure Initialize(var T: TAnyType; ACount: SizeInt)`

Visibility: default

Description: `Initialize` is a compiler intrinsic: it initializes a memory area belonging to a variable T of a managed type (1335) (`TManagedType`). Initializing means zeroing out the memory area. In this sense it is close in functionality to `Default` (1448), but `Default` returns an already initialized variable, whereas `Initialize` initializes managed types in a previously declared memory area.

`Default` is a function, and one can think of `Initialize` as the procedural version of `Default`.

`Initialize` performs the opposite operation of `Finalize` (1467), which should be used to clean up the memory block when it is no longer needed.

The optional `ACount` parameter can be used to initialize an array. It then specifies the number of elements in the array.

Note that the function can be used on an already used variable, and that for structured types, only managed fields are initialized.

See also: `Finalize` (1467), `Default` (1448), `ManagedTypes` (1335), `TypeInfo` (1563)

Listing: ./examples/refex/ex117.pp

```
{
  This example demonstrates the use of the Initialize and Finalize functions
  used to initialize (and clean up) any RTTI-enabled data not allocated with
  New or Create.
}

{$mode objfpc}
{$h+} // use ansistrings, they need to be initialized.
Type
  PData = ^TData;
  TData = record
    Street,City,Zip,Country, Tel: String;
    StreetNumber : Integer;
  end;

var
  Data: PData;

begin
  // Do not use New.
  GetMem(Data,SizeOf(TData));
  Try
    { Initialize the structure in memory, using Run-Time Type Information }
    Initialize(Data^);
    { Assign some string data to the ansistring contents.
      Note that this only works because the record was zeroed out by Initialize }
    Data^.Street:='Sesame Street';
    Data^.City:='Heaven';
    Data^.Zip:='7777777';
    Data^.Country:='Spain';
    Data^.StreetNumber:=3;
  Finally
    { Clean up the record contents.
      Again, the structure of the record is detected through
      Run-time Type Information }
    Finalize(Data^);
    FreeMem(Data);
  end;
end.
```

Listing: ./examples/refex/ex118.pp

```
{
  This example demonstrates the use of the Initialize and Finalize functions
  used to initialize (and clean up) any RTTI-enabled data not allocated with
```

```

    New or Create.
}

{$mode objfpc}
{$h+} // use ansistrings, they need to be initialized.
Type
  PData = ^TData;
  TData = record
    Street,City,Zip,Country, Tel: String;
    StreetNumber : Integer;
  end;

var
  Data: PData;

begin
  // We use the fact that a pointer is also usable as an array.
  GetMem(Data,SizeOf(TData)*2);
  Try
    { Initialize the structure in memory, using Run-Time Type Information}
    Initialize(Data^,2);
    { Assign some string data to the ansistring contents.
      Note that this only works because the record was zeroed out by Initialize}
    Data[0].Street:='Sesame Street';
    Data[0].City:='Heaven';
    Data[0].Zip:='7777777';
    Data[0].Country:='Spain';
    Data[0].StreetNumber:=3;
    // Second, well known street
    Data[1].Street:='Wall Street';
    Data[1].City:='New York';
    Data[1].Zip:='10005';
    Data[1].Country:='USA';
    Data[1].StreetNumber:=11;
  Finally
    { Clean up the record contents.
      Again, the structure of the record is detected through
      Run-time Type Information }
    Finalize(Data^);
    FreeMem(Data);
  end;
end.

```

Listing: ./examples/refex/ex123.pp

```

program ex123;

{
  Example to show that Initialize can be used on a variable that was already used,
  and that only fields of managed type are affected.
}

type
  TTest = object
    I: Integer;
    S: AnsiString;
  end;

procedure Test;

```

```

var
  V: TTest;

begin
  V.I := 10;
  V.S := 'x';
  WriteLn(V.I);
  WriteLn(V.S);
  Initialize(V);
  WriteLn(V.I); // unchanged
  WriteLn(V.S); // empty
end;

begin
  Test;
end.

```

75.12.183 InitializeArray

Synopsis: Initialize managed-type elements in array.

Declaration: `procedure InitializeArray(p: Pointer; typeInfo: Pointer; count: SizeInt)`

Visibility: default

Description: `InitializeArray` initializes managed types in the array pointed to by `p`. For this, it uses the type information of the elements as specified in `typeinfo`.

Under normal circumstances, this procedure should not be used, it is called automatically by the compiler when an array-typed variable is declared and the array contains elements with managed types.

See also: `FinalizeArray` ([1467](#)), `CopyArray` ([1446](#))

75.12.184 InitThread

Synopsis: Initialize a thread.

Declaration: `procedure InitThread(stklen: SizeUInt)`

Visibility: default

Description: Do not use, this is used internally by the thread manager.

75.12.185 InitThreadVars

Synopsis: Initialize threadvars.

Declaration: `procedure InitThreadVars(RelocProc: TRelocateThreadVarHandler)`

Visibility: default

Description: This routine should be called when threading is started. It is called by the compiler and should never be called manually, only from a thread manager.

Errors: None.

See also: `TThreadManager` ([1400](#))

75.12.186 Insert

Synopsis: Insert one string or dynamic array in another.

Declaration: `procedure Insert(const source: string; var S: string;
const Index: Integer)
procedure Insert(const source: DynArrayType; var S: DynArrayType;
const Index: Integer)`

Visibility: default

Description: `Insert` inserts string or character `Source` in string `S`, at position `Index`, shifting all characters after `Index` to the right. The resulting string is truncated at 255 characters, if needed. (i.e. for shortstrings)

`Index` is a 1-based index. if `Index` is less than 1, the insert of `Source` happens at the start of the string, as if the value 1 was specified.

If the value of `Index` is larger than the length of the string `S`, `Source` is appended to the string `S`.

For dynamic arrays, `Insert` inserts an element `Source` or the elements of array `Source` in array `S`, at position `Index`, shifting all elements after `Index` to the right.

`Index` is a 0-based index. if `Index` is less than 0, the insert of `Source` happens at position 0.

If the value of `Index` is larger than the length of the array `S`, `Source` is appended to the array `S`.

Errors: None.

See also: `Delete` ([1449](#)), `Copy` ([1445](#)), `Pos` ([1515](#))

Listing: `./examples/refex/ex33.pp`

Program Example33;

{ Program to demonstrate the Insert function. }

Var S : String;

begin

S:='Free Pascal is difficult to use !';

Insert ('NOT ',S,**pos**('difficult',S));

writeln (s);

end.

75.12.187 Int

Synopsis: Calculate integer part of floating point value.

Declaration: `function Int(d: ValReal) : ValReal`

Visibility: default

Description: `Int` returns the integer part of any Real X, as a Real.

Errors: None.

See also: `Frac` ([1471](#)), `Round` ([1529](#))

Listing: `./examples/refex/ex34.pp`

Program Example34;

{ Program to demonstrate the Int function. }

```
begin
  Writeln (Int(123.456):0:1); { Prints 123.0 }
  Writeln (Int(-123.456):0:1); { Prints -123.0 }
end.
```

75.12.188 intdivide(variant,variant):variant

Synopsis: Implement div (integer division) operation on variants.

Declaration: `operator div(const op1: variant; const op2: variant) : variant`

Visibility: default

Description: The implementation of the integer division Div operation is delegated to the variant manager with operation opintdivide.

Errors: Execution of this operator may result in an exception if no variant manager is installed or if the types of the operand are not suitable for the operation.

See also: operator mod(variant, variant): variant ([1335](#))

75.12.189 InterlockedCompareExchange

Synopsis: Conditional exchange.

```
Declaration: function InterlockedCompareExchange(var Target: LongInt;
          NewValue: LongInt;
          Comperand: LongInt) : LongInt
function InterlockedCompareExchange(var Target: Pointer;
          NewValue: Pointer;
          Comperand: Pointer) : Pointer
function InterlockedCompareExchange(var Target: Cardinal;
          NewValue: Cardinal;
          Comperand: Cardinal) : Cardinal
```

Visibility: default

Description: InterlockedCompareExchange does an compare-and-exchange operation on the specified values in a thread-safe way. The function compares Target and Comperand and exchanges Target with NewValue if Target and Comperand are equal. It returns the old value of Target. This is done in a thread-safe way, i.e., only one processor is accessing the Target variable at a time.

Errors: None.

See also: InterLockedDecrement ([1495](#)), InterLockedIncrement ([1497](#)), InterLockedExchange ([1495](#)), InterLockedExchangeAdd ([1496](#))

75.12.190 InterlockedCompareExchange64

Declaration: `function InterlockedCompareExchange64 (var Target: Int64;
 NewValue: Int64; Comperand: Int64)
 : Int64
 function InterlockedCompareExchange64 (var Target: QWord;
 NewValue: QWord; Comperand: QWord)
 : QWord`

Visibility: default

75.12.191 InterlockedCompareExchangePointer

Synopsis: Compare pointers in an atomic operation.

Declaration: `function InterlockedCompareExchangePointer (var Target: Pointer;
 NewValue: Pointer;
 Comperand: Pointer) : Pointer`

Visibility: default

Description: `InterlockedCompareExchangePointer` compares `Comperand` with `Target`. if they are equal, replaces `Target` with `NewValue`. This is done in a single atomic operation.

See also: `InterlockedCompareExchange` ([1494](#))

75.12.192 InterlockedDecrement

Synopsis: Thread-safe decrement.

Declaration: `function InterlockedDecrement (var Target: LongInt) : LongInt
 function InterlockedDecrement (var Target: Pointer) : Pointer
 function InterlockedDecrement (var Target: Cardinal) : Cardinal`

Visibility: default

Description: `InterLockedDecrement` decrements `Target` with 1 and returns the result. This is done in a thread-safe way. (i.e. only one processor is accessing the variable at a time).

Errors: None.

See also: `InterLockedIncrement` ([1497](#)), `InterLockedExchange` ([1495](#)), `InterLockedExchangeAdd` ([1496](#)), `InterlockedCompareExchange` ([1494](#))

75.12.193 InterlockedDecrement64

Declaration: `function InterlockedDecrement64 (var Target: Int64) : Int64
 function InterlockedDecrement64 (var Target: QWord) : QWord`

Visibility: default

75.12.194 InterlockedExchange

Synopsis: Exchange 2 integers in a thread-safe way.

Declaration: `function InterlockedExchange(var Target: LongInt; Source: LongInt)
: LongInt
function InterlockedExchange(var Target: Pointer; Source: Pointer)
: Pointer
function InterlockedExchange(var Target: Cardinal; Source: Cardinal)
: Cardinal`

Visibility: default

Description: `InterLockedExchange` stores `Source` in `Target` and returns the old value of `Target`. This is done in a thread-safe way, i.e., only one processor is accessing the `Target` variable at a time.

Errors: None.

See also: `InterLockedDecrement` ([1495](#)), `InterLockedIncrement` ([1497](#)), `InterLockedExchangeAdd` ([1496](#)), `InterlockedCompareExchange` ([1494](#))

75.12.195 InterlockedExchange64

Declaration: `function InterlockedExchange64(var Target: Int64; Source: Int64) : Int64
function InterlockedExchange64(var Target: QWord; Source: QWord) : QWord`

Visibility: default

75.12.196 InterlockedExchangeAdd

Synopsis: Thread-safe add and exchange of 2 values.

Declaration: `function InterlockedExchangeAdd(var Target: LongInt; Source: LongInt)
: LongInt
function InterlockedExchangeAdd(var Target: Pointer; Source: Pointer)
: Pointer
function InterlockedExchangeAdd(var Target: Cardinal; Source: Cardinal)
: Cardinal`

Visibility: default

Description: `InterLockedExchangeAdd` adds to `Target` the value of `Source` in a thread-safe way, and returns the old value of `Target`. This is done in a thread-safe way, i.e., only one processor is accessing the `Target` variable at a time.

Errors: None.

See also: `InterLockedDecrement` ([1495](#)), `InterLockedIncrement` ([1497](#)), `InterLockedExchange` ([1495](#)), `InterlockedCompareExchange` ([1494](#))

75.12.197 InterlockedExchangeAdd64

Declaration: `function InterlockedExchangeAdd64(var Target: Int64; Source: Int64)
: Int64
function InterlockedExchangeAdd64(var Target: QWord; Source: QWord)
: QWord`

Visibility: default

75.12.198 InterlockedIncrement

Synopsis: Thread-safe increment.

Declaration: `function InterlockedIncrement (var Target: LongInt) : LongInt`
`function InterlockedIncrement (var Target: Pointer) : Pointer`
`function InterlockedIncrement (var Target: Cardinal) : Cardinal`

Visibility: default

Description: `InterLockedIncrement` increments `Target` with 1 and returns the result. This is done in a thread-safe way (i.e. only one processor is accessing the variable at a time).

Errors: None.

See also: `InterLockedDecrement` ([1495](#)), `InterLockedExchange` ([1495](#)), `InterLockedExchangeAdd` ([1496](#)), `InterlockedCompareExchange` ([1494](#))

75.12.199 InterlockedIncrement64

Declaration: `function InterlockedIncrement64 (var Target: Int64) : Int64`
`function InterlockedIncrement64 (var Target: QWord) : QWord`

Visibility: default

75.12.200 IOResult

Synopsis: Return result of last file IO operation.

Declaration: `function IOResult : Word`

Visibility: default

Description: `IOresult` contains the result of any input/output call, when the `{\${i-}}` compiler directive is active, disabling IO checking. When the flag is read, it is reset to zero. If `IOresult` is zero, the operation completed successfully. If non-zero, an error occurred. The following errors can occur:

dos errors :

2 File not found.

3 Path not found.

4 Too many open files.

5 Access denied.

6 Invalid file handle.

12 Invalid file-access mode.

15 Invalid disk number.

16 Cannot remove current directory.

17 Cannot rename across volumes.

I/O errors :

100 Error when reading from disk.

101 Error when writing to disk.

102 File not assigned.

- 103 File not open.
- 104 File not opened for input.
- 105 File not opened for output.
- 106 Invalid number.

Fatal errors :

- 150 Disk is write protected.
- 151 Unknown device.
- 152 Drive not ready.
- 153 Unknown command.
- 154 CRC check failed.
- 155 Invalid drive specified..
- 156 Seek error on disk.
- 157 Invalid media type.
- 158 Sector not found.
- 159 Printer out of paper.
- 160 Error when writing to device.
- 161 Error when reading from device.
- 162 Hardware failure.

Errors: None.

Listing: ./examples/refex/ex35.pp

Program Example35;

{ Program to demonstrate the IOResult function. }

Var F : text;

begin

Assign (f,paramstr(1));

{ \$i- }

Reset (f);

{ \$i+ }

If IOresult<>0 **then**

writeln ('File ',paramstr(1),' doesn't exist')

else

writeln ('File ',paramstr(1),' exists');

end.

75.12.201 IsDynArrayRectangular

Synopsis: Check whether all dimensions have the same size.

Declaration: function IsDynArrayRectangular(a: Pointer; typeInfo: Pointer) : Boolean

Visibility: default

Description: IsDynArrayRectangular returns True if all dimensions of the dynamic array a with type information typinfo have the same bounds. It returns True if the array is empty.

See also: InitializeArray (1492), FinalizeArray (1467), CopyArray (1446), DynArraySize (1454), DynArrayClear (1453), DynArrayBounds (1453), DynArrayDim (1453)

75.12.202 IsMemoryManagerSet

Synopsis: Is the memory manager set.

Declaration: `function IsMemoryManagerSet : Boolean`

Visibility: default

Description: `IsMemoryManagerSet` will return `True` if the memory manager has been set to another value than the system heap manager, it will return `False` otherwise.

Errors: None.

See also: `SetMemoryManager` ([1539](#)), `GetMemoryManager` ([1475](#))

75.12.203 Is_IntResource

Synopsis: Check whether a resource is an internal resource.

Declaration: `function Is_IntResource(aStr: PChar) : Boolean`

Visibility: default

Description: `Is_IntResource` returns `True` if the resource type is internal (system predefined) resource or false if it is a user-defined resource type.

Errors: None.

75.12.204 KillThread

Synopsis: Kill a running thread.

Declaration: `function KillThread(threadHandle: TThreadID) : DWord`

Visibility: default

Description: `KillThread` causes a running thread to be aborted. The thread is identified by its handle or ID `threadHandle`.

The function returns zero if successful. A nonzero return value indicates failure.

Errors: If a failure occurred, a nonzero result is returned. The meaning is system dependent.

See also: `WaitForThreadTerminate` ([1572](#)), `EndThread` ([1455](#)), `SuspendThread` ([1555](#))

75.12.205 LazyInitThreading

Declaration: `procedure LazyInitThreading`

Visibility: default

75.12.206 LeaveCriticalSection

Synopsis: Leave a critical section.

Declaration: `procedure LeaveCriticalSection(var cs: TRTLCRITICALSECTION)`

Visibility: default

Description: `LeaveCriticalSection` signals that the current thread is exiting the critical section CS it has entered with `EnterCriticalSection` (1455).

The critical section must have been initialized with `InitCriticalSection` (1489) prior to a call to `EnterCriticalSection` and `LeaveCriticalSection`.

See also: `InitCriticalSection` (1489), `DoneCriticalSection` (1451), `EnterCriticalSection` (1455)

75.12.207 `leftshift(variant,variant):variant`

Synopsis: Implement binary `shl` operation on variants.

Declaration: `operator shl(const op1: variant; const op2: variant) : variant`

Visibility: default

Description: The implementation of the `shl` operation is delegated to the variant manager with operation `opshiftright`.

Errors: Execution of this operator may result in an exception if no variant manager is installed or if the types of the operand are not suitable for the operation.

See also: `operator shr(variant, variant): variant` (1335)

75.12.208 `Length`

Synopsis: Returns length of a string or array.

Declaration: `function &Length(S: AStringType) : SizeInt`
`function &Length(A: DynArrayType) : SizeInt`

Visibility: default

Description: `Length` returns the length of the string or array `S`, which is limited to 255 for shortstrings. If the string `S` is empty, 0 is returned.

Note: The length of the string `S` is stored in `S[0]` for shortstrings only. The `Length` function should always be used on ansistrings and widestrings.

For dynamic or static arrays, the function returns the number of elements in the array.

`Length` also supports arguments of type `PChar` and `PWideChar`, in which case it is identical to the `StrLen` and `WStrLen` functions, respectively. In this case, the function actually calculates the length of the null-terminated string, and its execution time is proportional to the string length because the terminating null character is searched through a linear scan.

Errors: None.

See also: `Pos` (1515), `SetLength` (1538)

Listing: `./examples/refex/ex36.pp`

Program Example36;

{ Program to demonstrate the Length function. }

type

 somebytes = **array** [6..10] **of** byte;
 somewords = **array** [3..10] **of** word;

```

Var
  S : String;
  I : Integer;
  bytes : somebytes;
  words : somewords;

begin
  S:="";
  for i:=1 to 10 do
    begin
      S:=S+' ';
      Writeln (Length(S):2, ' : ',s);
    end;
  Writeln('Bytes : ',length(bytes));
  Writeln('Words : ',length(words));
end.

```

75.12.209 **lessthan(variant,variant):Boolean**

Synopsis: Implement < (less than) operation on variants.

Declaration: `operator <(const op1: variant; const op2: variant) : Boolean`

Visibility: default

Description: The implementation of the "less than" comparison (<) operation is delegated to the variant manager with operation `opcmlt`.

Errors: Execution of this operator may result in an exception if no variant manager is installed or if the types of the operand are not suitable for the operation.

See also: operator `>(variant, variant): boolean` ([1335](#))

75.12.210 **lessthanorequal(variant,variant):Boolean**

Synopsis: Implement <= (less than or equal) operation on variants.

Declaration: `operator <=(const op1: variant; const op2: variant) : Boolean`

Visibility: default

Description: The implementation of the "less than or equal" comparison (<=) operation is delegated to the variant manager with operation `opcmlt`.

Errors: Execution of this operator may result in an exception if no variant manager is installed or if the types of the operand are not suitable for the operation.

See also: operator `<(variant, variant): boolean` ([1335](#))

75.12.211 **LEtoN**

Synopsis: Convert Little Endian-ordered integer to Native-ordered integer.

Declaration: `function LEtoN(const AValue: SmallInt) : SmallInt`
`function LEtoN(const AValue: Word) : Word`
`function LEtoN(const AValue: LongInt) : LongInt`

```
function LEtoN(const AValue: DWord) : DWord
function LEtoN(const AValue: Int64) : Int64
function LEtoN(const AValue: QWord) : QWord
```

Visibility: default

Description: `LEtoN` will rearrange the bytes in a Little-Endian number to the native order for the current processor. That is, for a little-endian processor, it will do nothing, and for a big-endian processor, it will invert the order of the bytes.

See also: `BEtoN` ([1430](#)), `NtoBE` ([1510](#)), `NtoLE` ([1510](#))

75.12.212 Ln

Synopsis: Calculate logarithm.

Declaration: `function Ln(d: ValReal) : ValReal`

Visibility: default

Description: `Ln` returns the natural logarithm of the Real parameter X. X must be positive.

Errors: An run-time error will occur when X is negative.

See also: `Exp` ([1462](#))

Listing: `./examples/refex/ex37.pp`

Program Example37;

{ Program to demonstrate the Ln function. }

```
begin
  Writeln (Ln(1)); { Prints 0 }
  Writeln (Ln(Exp(1))); { Prints 1 }
end.
```

75.12.213 Lo

Synopsis: Return low nibble/byte/word of value.

Declaration: `function Lo(B: Byte) : Byte`
`function Lo(i: Integer) : Byte`
`function Lo(w: Word) : Byte`
`function Lo(l: LongInt) : Word`
`function Lo(l: DWord) : Word`
`function Lo(i: Int64) : DWord`
`function Lo(q: QWord) : DWord`

Visibility: default

Description: `Lo` returns the low nibble, byte or word or longword from X, depending on the size of X.

Table 75.22:

Size	Return value
8	Byte, low nibble
16	Byte, low byte
32	Word, low word
64	Cardinal, low DWord

Errors: None.

See also: Ord ([1512](#)), Chr ([1437](#)), Hi ([1482](#))

Listing: ./examples/refex/ex38.pp

Program Example38;

{ Program to demonstrate the Lo function. }

```

Var L : Longint;
      W : Word;
      B : Byte;
begin
  L:=(1 Shl 16) + (1 Shl 4); { $10010 }
  Writeln (Lo(L)); { Prints 16 }
  W:=(1 Shl 8) + (1 Shl 4); { $110 }
  Writeln (Lo(W)); { Prints 16 }
  B:=$EF;
  Writeln (Lo(B)); { Prints 15 }
end.

```

75.12.214 LoadLibrary

Synopsis: Load a dynamic library and return a handle to it.

Declaration: `function LoadLibrary(const Name: RawByteString) : TLibHandle`
`function LoadLibrary(const Name: UnicodeString) : TLibHandle`

Visibility: default

Description: LoadLibrary loads a dynamic library in file Name and returns a handle to it. If the library cannot be loaded, NilHandle ([1359](#)) is returned.

No assumptions should be made about the location of the loaded library if a relative pathname is specified. The behaviour is dependent on the platform. Therefore it is best to specify an absolute pathname if possible.

Errors: On error, NilHandle ([1359](#)) is returned.

See also: UnloadLibrary ([1566](#)), GetProcAddress ([1476](#))

75.12.215 LoadResource

Synopsis: Load a resource for use.

Declaration: `function LoadResource (ModuleHandle: TFPResourceHMODULE;
ResHandle: TFPResourceHandle) : TFPResourceHGLOBAL`

Visibility: default

Description: `LoadResource` loads a resource identified by `ResHandle` from a module identified by `ModuleHandle` into memory. It returns a handle to the resource.

Loaded resources must be unloaded again using the `FreeResource` (1472) function.

Errors: On error, 0 is returned.

See also: `FindResource` (1468), `FreeResource` (1472), `SizeofResource` (1546), `LockResource` (1504), `UnlockResource` (1567), `FreeResource` (1472)

75.12.216 LockResource

Synopsis: Lock a resource.

Declaration: `function LockResource (ResData: TFPResourceHGLOBAL) : Pointer`

Visibility: default

Description: `LockResource` locks a resource previously loaded by `LoadResource` into memory. This means that any attempt to modify the resource will fail while it is locked. The function returns a pointer to the resource location in memory.

The resource can be freed again using the `UnlockResource` (1567) function.

Errors: if the function fails, `Nil` is returned.

See also: `FindResource` (1468), `FreeResource` (1472), `SizeofResource` (1546), `LoadResource` (1503), `UnlockResource` (1567), `FreeResource` (1472)

75.12.217 logicaland(variant,variant):variant

Synopsis: Implement logical/binary and operation on variants.

Declaration: `operator and (const op1: variant; const op2: variant) : variant`

Visibility: default

Description: The implementation of the `and` operation is delegated to the variant manager with operation `opand`.

Errors: Execution of this operator may result in an exception if no variant manager is installed or if the types of the operand are not suitable for the operation.

See also: `operator or (variant, variant): variant` (1335), `operator xor (variant, variant): variant` (1335), `operator not (variant): variant` (1335)

75.12.218 logicalnot(variant):variant

Synopsis: Implement logical/binary not operation on variants.

Declaration: `operator not (const op: variant) : variant`

Visibility: default

Description: The implementation of the `not` operation is delegated to the variant manager with operation `opnot`.

Errors: Execution of this operator may result in an exception if no variant manager is installed or if the types of the operand are not suitable for the operation.

See also: operator `and(variant, variant): variant` ([1335](#)), operator `or(variant, variant): variant` ([1335](#)), operator `xor(variant, variant): variant` ([1335](#))

75.12.219 `logicalor(variant,variant):variant`

Synopsis: Implement logical/binary `or` operation on variants.

Declaration: `operator or(const op1: variant; const op2: variant) : variant`

Visibility: default

Description: The implementation of the `or` operation is delegated to the variant manager with operation `opor`.

Errors: Execution of this operator may result in an exception if no variant manager is installed or if the types of the operand are not suitable for the operation.

See also: operator `and(variant, variant): variant` ([1335](#)), operator `xor(variant, variant): variant` ([1335](#)), operator `not(variant): variant` ([1335](#))

75.12.220 `logicalxor(variant,variant):variant`

Synopsis: Implement logical/binary `xor` operation on variants.

Declaration: `operator xor(const op1: variant; const op2: variant) : variant`

Visibility: default

Description: The implementation of the `xor` operation is delegated to the variant manager with operation `opxor`.

Errors: Execution of this operator may result in an exception if no variant manager is installed or if the types of the operand are not suitable for the operation.

See also: operator `or(variant, variant): variant` ([1335](#)), operator `and(variant, variant): variant` ([1335](#)), operator `not(variant): variant` ([1335](#))

75.12.221 `longjmp`

Synopsis: Jump to address.

Declaration: `procedure longjmp(var S: jmp_buf; value: LongInt)`

Visibility: default

Description: `LongJump` jumps to the address in `S` (a `jmp_buf`), and restores the registers that were stored in it at the corresponding `SetJump` ([1537](#)) call. In effect, program flow will continue at the `SetJump` call, which will return `value` instead of 0. If a value equal to zero is passed, it will be converted to 1 before passing it on. The call will not return, so it must be used with extreme care. This can be used for error recovery, for instance when a segmentation fault occurred.

For an example, see `SetJump` ([1537](#))

Errors: None.

See also: `SetJump` ([1537](#))

75.12.222 Low

Synopsis: Return lowest index of open array or enumerated.

Declaration: `function Low(Arg: TypeOrVariable) : TOrdinal`

Visibility: default

Description: The return value of `Low` depends on it's argument:

- 1.If the argument is an ordinal type, `Low` returns the lowest value in the range of the given ordinal type.
- 2.If the argument is an array type or an array type variable then `Low` returns the lowest possible value of it's index.
- 3.If the argument is an open array identifier in a function or procedure, then `Low` returns the lowest element of the array, which is always zero.
- 4.If the argument is a set type then it returns the lowest value of the underlying ordinal type.

The return type is always the same type as the type of the argument.

for an example, see `High` ([1483](#)).

Errors: None.

See also: `High` ([1483](#)), `Ord` ([1512](#)), `Pred` ([1517](#)), `Succ` ([1554](#))

75.12.223 LowerCase

Synopsis: Return lowercase version of a string.

Declaration: `function LowerCase(const s: shortstring) : shortstring; Overload`
`function LowerCase(c: Char) : Char; Overload`
`function LowerCase(const s: ansistring) : ansistring`
`function LowerCase(const s: UnicodeString) : UnicodeString`
`function LowerCase(c: UnicodeChar) : UnicodeChar`

Visibility: default

Description: `Lowercase` returns the lowercase version of its argument `C`. If its argument is a string, then the complete string is converted to lowercase. The type of the returned value is the same as the type of the argument.

`Lowercase` does not change the number of characters (or bytes) in an `ansistring`.

Errors: None.

See also: `Uppcase` ([1567](#))

Listing: `./examples/refex/ex73.pp`

```

program Example73;

{ Program to demonstrate the Lowercase function. }

var c:char;

begin
  for c:='A' to 'Z' do
    write(lowercase(c));

```

```

Writeln;
Writeln(Lowercase('ABCDEFGHIJKLMNOPQRSTUVWXYZ'));
end.

```

75.12.224 MakeLangID

Synopsis: Create a language ID.

Declaration: `function MakeLangID(primary: Word; sub: Word) : Word`

Visibility: default

Description: `MakeLangID` creates a language ID from the `primary` and `sub` language IDs.

75.12.225 MemSize

Synopsis: Return the size of a memory block.

Declaration: `function MemSize(p: pointer) : PtrUInt`

Visibility: default

Description: `MemSize` returns the size of a memory block on the heap.

Errors: Passing an invalid pointer may lead to run-time errors (access violations).

See also: `GetMem` ([1474](#)), `FreeMem` ([1471](#))

75.12.226 MkDir

Synopsis: Create a new directory.

Declaration: `procedure MkDir(const s: shortstring); Overload`
`procedure MkDir(const s: RawByteString); Overload`
`procedure MkDir(const s: unicodestring); Overload`

Visibility: default

Description: `Mkdir` creates a new directory `S`.

For an example, see `Rmdir` ([1526](#)).

Errors: Depending on the state of the `{ $I }` switch, a runtime error can be generated if there is an error. In the `{ $I- }` state, use `IOResult` to check for errors.

See also: `Chdir` ([1436](#)), `Rmdir` ([1526](#))

75.12.227 modulus(variant,variant):variant

Synopsis: Implement `mod` (modulo) operation on variants.

Declaration: `operator mod(const op1: variant; const op2: variant) : variant`

Visibility: default

Description: The implementation of the modulo `Mod` operation is delegated to the variant manager with operation `opModulus`.

Errors: Execution of this operator may result in an exception if no variant manager is installed or if the types of the operand are not suitable for the operation.

See also: operator `div(variant, variant)`: variant ([1335](#))

75.12.228 Move

Synopsis: Move data from one location in memory to another.

Declaration: `procedure Move(const source; var dest; count: SizeInt)`

Visibility: default

Description: `Move` moves `Count` bytes from `Source` to `Dest`.

Errors: If either `Dest` or `Source` is outside the accessible memory for the process, then a run-time error will be generated.

See also: `Fillword` ([1466](#)), `Fillchar` ([1465](#))

Listing: `./examples/refex/ex42.pp`

Program Example42;

{ Program to demonstrate the Move function. }

Var S1,S2 : **String** [30];

begin

 S1:='Hello World !';

 S2:='Bye, bye !';

Move (S1,S2,**Sizeof**(S1));

Writeln (S2);

end.

75.12.229 MoveChar0

Synopsis: Move data till first zero character.

Declaration: `procedure MoveChar0(const buf1; var buf2; len: SizeInt)`

Visibility: default

Description: `MoveChar0` moves `Count` bytes from `buf1` to `buf2`, and stops moving if a zero character is found.

Errors: No checking is done to see if `Count` stays within the memory allocated to the process.

See also: `Move` ([1508](#))

Listing: `./examples/refex/ex109.pp`

Program Example109;

{ Program to demonstrate the MoveChar0 function. }

Var

 Buf1,Buf2 : **Array**[1..80] **of** char;

```

I : longint;

begin
  Randomize;
  For I:=low(buf1) to high(buf1) do
    Buf1[i]:=chr(Random(16)+Ord('A'));
  Writeln('Original buffer');
  writeln(Buf1);
  Buf1[Random(80)+1]:=#0;
  MoveChar0(Buf1,Buf2,80);
  Writeln('Randomly zero-terminated Buffer');
  Writeln(Buf2);
end.

```

75.12.230 multiply(variant,variant):variant

Synopsis: Implement multiplication (`*`) operation on variants.

Declaration: `operator *(const op1: variant; const op2: variant) : variant`

Visibility: default

Description: The implementation of the multiplication `*` operation is delegated to the variant manager with operation `opMultiply`.

Errors: Execution of this operator may result in an exception if no variant manager is installed or if the types of the operand are not suitable for the operation.

See also: `operator /(variant, variant): variant` ([1335](#))

75.12.231 negative(variant):variant

Synopsis: Implement `-` (unary minus, negation) operation on variants.

Declaration: `operator -(const op: variant) : variant`

Visibility: default

Description: The implementation of the unary minus (`-`) operation is delegated to the variant manager with operation `varNeg`.

Errors: Execution of this operator may result in an exception if no variant manager is installed or if the types of the operand are not suitable for the operation.

See also: `operator -(variant, variant): variant` ([1335](#))

75.12.232 New

Synopsis: Dynamically allocate memory for variable.

Declaration: `procedure &New(var P: Pointer)`
`procedure &New(var P: Pointer; Cons: TProcedure)`

Visibility: default

Description: `New` allocates a new instance of the type pointed to by `P`, and puts the address in `P`. If `P` is an object, then it is possible to specify the name of the constructor with which the instance will be created.

The newly allocated memory is not initialized in any way, and may contain garbage data. It must be cleared with a call to `FillChar` (1465) or `FillWord` (1466).

For an example, see `Dispose` (1450).

Errors: What happens if no more memory is available, depends on the value of the variable `ReturnNilIfGrowHeapfails` (1417): if the variable is `True` then `Nil` is returned. If the variable is `False`, a run-time error is generated.

See also: `Dispose` (1450), `Freemem` (1471), `Getmem` (1474), `ReturnNilIfGrowHeapfails` (1417)

75.12.233 NtoBE

Synopsis: Convert Native-ordered integer to a Big Endian-ordered integer.

Declaration:

```
function NtoBE(const AValue: SmallInt) : SmallInt
function NtoBE(const AValue: Word) : Word
function NtoBE(const AValue: LongInt) : LongInt
function NtoBE(const AValue: DWord) : DWord
function NtoBE(const AValue: Int64) : Int64
function NtoBE(const AValue: QWord) : QWord
```

Visibility: default

Description: `NtoBE` will rearrange the bytes in a natively-ordered number to the Big-Endian order. That is, for a Little-Endian processor, it will invert the order of the bytes and for a big-endian processor, it will do nothing.

See also: `BEtoN` (1430), `LEtoN` (1501), `NtoLE` (1510)

75.12.234 NtoLE

Synopsis: Convert Native-ordered integer to a Little Endian-ordered integer.

Declaration:

```
function NtoLE(const AValue: SmallInt) : SmallInt
function NtoLE(const AValue: Word) : Word
function NtoLE(const AValue: LongInt) : LongInt
function NtoLE(const AValue: DWord) : DWord
function NtoLE(const AValue: Int64) : Int64
function NtoLE(const AValue: QWord) : QWord
```

Visibility: default

Description: `NtoLE` will rearrange the bytes in a natively-ordered number to the little-Endian order. That is, for a Big-Endian processor, it will invert the order of the bytes and for a Little-Endian processor, it will do nothing.

See also: `BEtoN` (1430), `LEtoN` (1501), `NtoBE` (1510)

75.12.235 Null

Synopsis: Null variant.

Declaration:

```
function Null : Variant
```

Visibility: default

75.12.236 OctStr

Synopsis: Convert integer to a string with octal representation.

Declaration: `function OctStr(Val: LongInt; cnt: Byte) : shortstring`
`function OctStr(Val: Int64; cnt: Byte) : shortstring`
`function OctStr(Val: QWord; cnt: Byte) : shortstring`

Visibility: default

Description: `OctStr` returns a string with the octal representation of `Value`. The string has exactly `cnt` characters.

Errors: None.

See also: `Str` ([1549](#)), `Val` ([1570](#)), `BinStr` ([1430](#)), `HexStr` ([1481](#))

Listing: `./examples/refex/ex112.pp`

Program `example112;`

{ Program to demonstrate the OctStr function }

Const `Value = 45678;`

Var `l : longint;`

begin

For `l:=1 to 10 do`

Writeln (`OctStr(Value,l)`);

For `l:=1 to 16 do`

Writeln (`OctStr(l,3)`);

end.

75.12.237 Odd

Synopsis: Is a value odd or even ?

Declaration: `function Odd(l: LongInt) : Boolean`
`function Odd(l: LongWord) : Boolean`
`function Odd(l: Int64) : Boolean`
`function Odd(l: QWord) : Boolean`

Visibility: default

Description: `Odd` returns `True` if `X` is odd, or `False` otherwise.

Errors: None.

See also: `Abs` ([1418](#)), `Ord` ([1512](#))

Listing: `./examples/refex/ex43.pp`

Program `Example43;`

{ Program to demonstrate the Odd function. }

begin

If `Odd(1)` **Then**

```

    Writeln ('Everything OK with 1 !');
  If Not Odd(2) Then
    Writeln ('Everything OK with 2 !');
end.

```

75.12.238 Ofs

Synopsis: Return offset of a variable.

Declaration: `function Ofs(var X) : LongInt`

Visibility: default

Description: `Ofs` returns the offset of the address of a variable. This function is only supported for compatibility. In Free Pascal, it returns always the complete address of the variable, since Free Pascal is a 32/64 bit compiler.

Errors: None.

See also: `DSeg` ([1452](#)), `CSeg` ([1447](#)), `Seg` ([1536](#)), `Ptr` ([1517](#))

Listing: `./examples/refex/ex44.pp`

Program Example44;

{ Program to demonstrate the Ofs function. }

Var W : Pointer;

begin

 W:=Pointer(**Ofs**(W)); *{ W contains its own offset. }*

end.

75.12.239 Ord

Synopsis: Return ordinal value of an ordinal type.

Declaration: `function Ord(X: TOrdinal) : LongInt`

Visibility: default

Description: `Ord` returns the Ordinal value of a ordinal-type variable X.

Historical note:

Originally, Pascal did not have typecasts and `ord` was a necessary function in order to do certain operations on non-integer ordinal types. With the arrival of typecasting a generic approach became possible, making `ord` mostly obsolete. However `ord` is not considered deprecated and remains in wide use today.

Errors: None.

See also: `Chr` ([1437](#)), `Succ` ([1554](#)), `Pred` ([1517](#)), `High` ([1483](#)), `Low` ([1506](#))

Listing: `./examples/refex/ex45.pp`

Program Example45;

{ Program to demonstrate the Ord, Pred, Succ functions. }

Type

TEnum = (Zero, One, Two, Three, Four);

Var

X : Longint;

Y : TEnum;

begin

X:=125;

Writeln (Ord(X)); *{ Prints 125 }*

X:=Pred(X);

Writeln (Ord(X)); *{ prints 124 }*

Y:= One;

Writeln (Ord(y)); *{ Prints 1 }*

Y:=Succ(Y);

Writeln (Ord(Y)); *{ Prints 2 }*

end.

75.12.240 Pack

Synopsis: Create packed array from normal array.

Declaration: `procedure Pack(const A: UnpackedArrayType; StartIndex: TIndexType;
out Z: PackedArrayType)`

Visibility: default

Description: Pack will copy the elements of an unpacked array (A) to a packed array (Z). It will start the copy at the index denoted by StartIndex. The type of the index variable StartIndex must match the type of the index of A. The elements are always transferred to the beginning of the packed array Z. (i.e. it starts at Low(Z)).

Obviously, the type of the elements of the arrays A and Z must match.

See also: `unpack` ([1567](#))

75.12.241 ParamCount

Synopsis: Return number of command-line parameters passed to the program.

Declaration: `function ParamCount : LongInt`

Visibility: default

Description: Paramcount returns the number of command-line arguments. If no arguments were given to the running program, 0 is returned.

Errors: None.

See also: `Paramstr` ([1514](#))

Listing: `./examples/refex/ex46.pp`

Program Example46;

```

{ Program to demonstrate the ParamCount and ParamStr functions. }
Var
  I : LongInt;

begin
  Writeln (paramstr(0),' : Got ',ParamCount,' command-line parameters: ');
  For i:=1 to ParamCount do
    Writeln (ParamStr (i));
end.

```

75.12.242 ParamStr

Synopsis: Return value of a command-line argument.

Declaration: `function ParamStr(l: LongInt) : string`

Visibility: default

Description: `ParamStr` returns the `L`-th command-line argument. `L` must be between 0 and `Paramcount`, these values included. The zeroth argument is the path and file name with which the program was started.

The command-line parameters will be truncated to a length of 255, even though the operating system may support bigger command-lines. The `Objpas` unit (used in `objfpc` or `delphi` mode) defines versions of `ParamStr` which return the full-length command-line arguments, using `ansistrings`.

In the interest of portability, the `ParamStr` function tries to behave the same on all operating systems: like the original `ParamStr` function in Turbo Pascal. This means even on Unix, `paramstr(0)` returns the full path to the program executable. A notable exception is Mac OS X, where the returned value depends on how the application was started. It may be that just the name of the application is returned (in case of a command-line launch), so it is best to avoid using it on that platform.

In general, it's a bad idea to rely on the location of the binary. Often, this goes against best OS practices. Configuration data should (or can) not be stored next to the binary, but on designated locations. What locations these are, is very much operating system dependent. Therefore, `ParamStr(0)` should be used with care.

For an example, see `Paramcount` ([1513](#)).

Errors: None.

See also: `Paramcount` ([1513](#))

75.12.243 Pi

Synopsis: Return the value of PI.

Declaration: `function Pi : ValReal`

Visibility: default

Description: `Pi` returns the value of Pi (3.1415926535897932385).

Errors: None.

See also: `Cos` ([1446](#)), `Sin` ([1545](#))

Listing: ./examples/refex/ex47.pp

Program Example47;

{ Program to demonstrate the Pi function. }

```
begin
  Writeln (Pi); {3.1415926}
  Writeln (Sin(Pi));
end.
```

75.12.244 PopCnt

Synopsis: Count number of set bits.

Declaration: function PopCnt(const AValue: Byte) : Byte
 function PopCnt(const AValue: Word) : Word
 function PopCnt(const AValue: DWord) : DWord
 function PopCnt(const AValue: QWord) : QWord

Visibility: default

Description: PopCnt (population count) counts the number of set bits in AValue.

75.12.245 Pos

Synopsis: Search for substring in a string.

Declaration: function Pos(const substr: shortstring; const s: shortstring;
 Offset: SizeInt) : SizeInt
 function Pos(C: Char; const s: shortstring; Offset: SizeInt) : SizeInt
 function Pos(const Substr: ShortString; const Source: RawByteString;
 Offset: SizeInt) : SizeInt
 function Pos(const substr: shortstring; c: Char; Offset: SizeInt)
 : SizeInt
 function Pos(const Substr: RawByteString; const Source: RawByteString;
 Offset: SizeInt) : SizeInt
 function Pos(c: AnsiChar; const s: RawByteString; Offset: SizeInt)
 : SizeInt
 function Pos(const Substr: UnicodeString; const Source: UnicodeString;
 Offset: SizeInt) : SizeInt
 function Pos(c: Char; const s: UnicodeString; Offset: SizeInt) : SizeInt
 function Pos(c: UnicodeChar; const s: UnicodeString; Offset: SizeInt)
 : SizeInt
 function Pos(const c: RawByteString; const s: UnicodeString;
 Offset: SizeInt) : SizeInt
 function Pos(const c: UnicodeString; const s: RawByteString;
 Offset: SizeInt) : SizeInt
 function Pos(const c: ShortString; const s: UnicodeString;
 Offset: SizeInt) : SizeInt
 function Pos(const Substr: WideString; const Source: WideString;
 Offset: SizeInt) : SizeInt
 function Pos(c: Char; const s: WideString; Offset: SizeInt) : SizeInt
 function Pos(c: WideChar; const s: WideString; Offset: SizeInt)

```

        : SizeInt
function Pos(c: WideChar; const s: RawByteString; Offset: SizeInt)
        : SizeInt
function Pos(const c: RawByteString; const s: WideString;
        Offset: SizeInt) : SizeInt
function Pos(const c: WideString; const s: RawByteString;
        Offset: SizeInt) : SizeInt
function Pos(const c: ShortString; const s: WideString; Offset: SizeInt)
        : SizeInt
function Pos(c: Char; const v: Variant) : SizeInt
function Pos(s: ShortString; const v: Variant) : SizeInt
function Pos(const a: AnsiString; const v: Variant) : SizeInt
function Pos(const w: WideString; const v: Variant) : SizeInt
function Pos(const w: UnicodeString; const v: Variant) : SizeInt
function Pos(const v: Variant; const c: Char) : SizeInt
function Pos(const v: Variant; const s: ShortString) : SizeInt
function Pos(const v: Variant; const a: AnsiString) : SizeInt
function Pos(const v: Variant; const w: WideString) : SizeInt
function Pos(const v: Variant; const w: UnicodeString) : SizeInt
function Pos(const v1: Variant; const v2: Variant) : SizeInt

```

Visibility: default

Description: Pos returns the index of Substr (or character c) in S, if S contains Substr. In case Substr isn't found, 0 is returned. The search is case-sensitive. The character version uses an optimized version of the search algorithm.

In case Offset is specified, the search starts at position Offset (1-based) in the string; The position Offset is also checked in the search.

Errors: None

See also: Length ([1500](#)), Copy ([1445](#)), Delete ([1449](#)), Insert ([1493](#)), posEx ([1321](#))

Listing: ./examples/refex/ex48.pp

Program Example48;

{ Program to demonstrate the Pos function. }

Var

S : String;

begin

S:='The first space in this sentence is at position : ';

Writeln (S,pos(' ',S));

S:='The last letter of the alphabet doesn't appear in this sentence ';

If (Pos ('Z',S)=0) **and** (Pos('z',S)=0) **then**

Writeln (S);

end.

75.12.246 power(variant,variant):variant

Synopsis: Implement power (**) operation on variants.

Declaration: operator ** (const op1: variant; const op2: variant) : variant

Visibility: default

Description: The implementation of the power `**` operation is delegated to the variant manager with operation `opPower`.

Errors: Execution of this operator may result in an exception if no variant manager is installed or if the types of the operand are not suitable for the operation.

See also: operator `*(variant, variant): variant` ([1335](#))

75.12.247 Pred

Synopsis: Return previous element for an ordinal type.

Declaration: `function Pred(X: TOrdinal) : TOrdinal`

Visibility: default

Description: `Pred` returns the element that precedes the element that was passed to it. If it is applied to the first value of the ordinal type, and the program was compiled with range checking on (`{ $R+ }`), then a run-time error will be generated.

For an example, see `Ord` ([1512](#))

You cannot use this function on enumerations with assigned values.

Errors: Run-time error 201 is generated when the result is out of range.

See also: `Ord` ([1512](#)), `Succ` ([1554](#)), `High` ([1483](#)), `Low` ([1506](#))

75.12.248 Prefetch

Synopsis: Prefetch a memory location.

Declaration: `procedure Prefetch(const mem)`

Visibility: default

Description: `Prefetch` can be used to optimize the CPU behaviour by already loading a memory location. It is mainly used as a hint for those processors that support it.

Errors: None.

75.12.249 Ptr

Synopsis: Combine segment and offset to pointer.

Declaration: `function Ptr(sel: LongInt; off: LongInt) : FarPointer`

Visibility: default

Description: `Ptr` returns a pointer, pointing to the address specified by segment `Sel` and offset `Off`.

Remark

1. In the 32/64-bit flat-memory model supported by Free Pascal, this function is obsolete.
2. The returned address is simply the offset.

Errors: None.

See also: [Addr \(1420\)](#)

Listing: ./examples/refex/ex59.pp

Program Example59;

{ Program to demonstrate the Ptr (compatibility) function. }

type pString = ^**String**;

Var P : pString;
 S : **String**;

begin
 S:='Hello, World !';
 P:=pString(**Ptr**(**Seg**(S),Longint(**Ofs**(S))));
 {P now points to S !}
 Writeln (P^);
end.

75.12.250 RaiseList

Synopsis: List of currently raised exceptions.

Declaration: function RaiseList : PExceptObject

Visibility: default

Description: RaiseList returns a pointer to the list of currently raised exceptions (i.e. a pointer to the first exception block).

75.12.251 Random

Synopsis: Generate random number.

Declaration: function Random(l: LongInt) : LongInt
 function Random(l: Int64) : Int64
 function Random : extended

Visibility: default

Description: Random returns a random number larger or equal to 0 and strictly less than L. For negative values of L the behaviour is undefined. If the argument L is omitted, a Real number between 0 and 1 is returned (0 included, 1 excluded).

Remark The Free Pascal implementation of the Random routine uses a Mersenne Twister algorithm to simulate randomness. This implementation has a better statistical distribution than for example a Linear Congruential generator algorithm, but is considerably slower than the latter. If speed is an issue, then alternate random number generators should be considered.

Note that the fact that a Mersenne Twister is used is an implementation detail, which can be changed at any point. The only guarantee Random() offers is that setting randseed to particular value will result in the same sequence of random numbers in a particular version the RTL. A newer version of the RTL may result in a different sequence for the same randseed.

Errors: None.

See also: [Randomize \(1519\)](#)

Listing: ./examples/refex/ex49.pp

Program Example49;

{ Program to demonstrate the Random and Randomize functions. }

Var I,Count,guess : Longint;
R : Real;

begin

Randomize; *{ This way we generate a new sequence every time
the program is run }*

Count:=0;

For i:=1 **to** 1000 **do**

If Random>0.5 **then inc**(Count);

Writeln ('Generated ',Count,' numbers > 0.5');

Writeln ('out of 1000 generated numbers.');

count:=0;

For i:=1 **to** 5 **do**

begin

write ('Guess a number between 1 and 5 : ');

readln(Guess);

If Guess=Random(5)+1 **then inc**(count);

end;

Writeln ('You guessed ',Count,' out of 5 correct.');

end.

75.12.252 Randomize

Synopsis: Initialize random number generator.

Declaration: `procedure Randomize`

Visibility: default

Description: `Randomize` initializes the random number generator of Free Pascal, by giving a value to `Randseed`, calculated with the system clock.

For an example, see `Random` ([1518](#)).

Errors: None.

See also: `Random` ([1518](#))

75.12.253 Read

Synopsis: Read from a text file into variable.

Declaration: `procedure Read(var F: Text; Args: Arguments)`
`procedure Read(Args: Arguments)`

Visibility: default

Description: `Read` reads one or more values from a file `F`, and stores the result in `V1`, `V2`, etc.; If no file `F` is specified, then standard input is read. If `F` is of type `Text`, then the variables `V1`, `V2` etc. must be of type `Char`, `Integer`, `Real`, `String`. If `F` is a typed file, then each of the variables must be of the type specified in the declaration of `F`. Untyped files are not allowed as an argument.

In earlier versions of FPC, it was also allowed to read `Pchar` null-terminated strings, but this has been removed, since there is no buffer checking possible.

Errors: If no data is available, empty values are returned (0 for ordinal values, empty strings for string values)

See also: [ReadLn \(1521\)](#), [Blockread \(1431\)](#), [Write \(1574\)](#), [Blockwrite \(1432\)](#)

Listing: ./examples/refex/ex50.pp

Program Example50;

{ Program to demonstrate the Read(Ln) function. }

Var S : **String**;

 C : **Char**;

 F : **File of char**;

begin

 Assign (F,'ex50.pp');

Reset (F);

 C:='A';

Writeln ('The characters before the first space in ex50.pp are : ');

While not Eof(f) and (C<>' ') do

Begin

Read (F,C);

Write (C);

end;

Writeln;

 Close (F);

Writeln ('Type some words. An empty line ends the program.');

repeat

ReadLn (S);

until S="";

end.

75.12.254 ReadBarrier

Synopsis: Memory Read Barrier.

Declaration: `procedure ReadBarrier`

Visibility: default

Description: `ReadBarrier` is a low-level instruction to force a read barrier in the CPU: all memory reads before the instruction will be finished before this instruction, before memory reads after the instruction occur.

See also: [ReadDependencyBarrier \(1520\)](#), [ReadWriteBarrier \(1522\)](#), [WriteBarrier \(1575\)](#)

75.12.255 ReadDependencyBarrier

Synopsis: Memory Read Dependency Barrier.

Declaration: `procedure ReadDependencyBarrier`

Visibility: default

Description: `ReadDependencyBarrier` is a low-level instruction to force a read barrier in the CPU: all memory reads (loads) depending on previous loads are separate from the ones following the instruction.

See also: [ReadBarrier \(1520\)](#), [ReadWriteBarrier \(1522\)](#), [WriteBarrier \(1575\)](#)

75.12.256 ReadLn

Synopsis: Read from a text file into variable and goto next line.

Declaration: `procedure ReadLn (var F: Text; Args: Arguments)`
`procedure ReadLn (Args: Arguments)`

Visibility: default

Description: `Read` reads one or more values from a file `F`, and stores the result in `V1`, `V2`, etc. After that it goes to the next line in the file. The end of the line is marked by any of the supported line ending styles, independent of the platform on which the code is running (supported line ending styles are CRLF, LF or CR). The end-of-line marker is not considered part of the line and is ignored.

If no file `F` is specified, then standard input is read. The variables `V1`, `V2` etc. must be of type `Char`, `Integer`, `Real`, `String` or `PChar`.

For an example, see `Read` ([1519](#)).

Errors: If no data is available, empty values are returned (0 for ordinal values, empty strings for string values)

See also: `Read` ([1519](#)), `Blockread` ([1431](#)), `Write` ([1574](#)), `Blockwrite` ([1432](#))

75.12.257 ReadStr

Synopsis: Read variables from a string.

Declaration: `procedure ReadStr (const S: string; Args: Arguments)`

Visibility: default

Description: `ReadStr` behaves like `Read` ([1519](#)), except that it reads its input from the string variable `S` instead of a file. Semantically, the `ReadStr` call is equivalent to writing the string to a file using the `Write` call, and then reading them into the various arguments `Arg` using the `Read` call from the same file:

```
var
  F : Text;
begin
  Rewrite (F);
  Write (F, S);
  Close (F);
  Reset (F);
  Read (F, Args);
  Close (F);
end;
```

Obviously, the `ReadStr` call does not use a temporary file.

`ReadStr` is defined in the ISO Extended Pascal standard. More information on the allowed arguments and the behaviour of the arguments can be found in the description of `Read` ([1519](#)).

See also: `Read` ([1519](#)), `WriteStr` ([1576](#)), `Write` ([1574](#))

75.12.258 ReadWriteBarrier

Synopsis: Memory read/write barrier.

Declaration: `procedure ReadWriteBarrier`

Visibility: default

Description: `ReadWriteBarrier` is a low-level instruction to force a read/write barrier in the CPU: both read (Loads) and write (stores) operations before and after the barrier are separate.

See also: `ReadBarrier` ([1520](#)), `ReadDependencyBarrier` ([1520](#)), `WriteBarrier` ([1575](#))

75.12.259 Real2Double

Synopsis: Convert Turbo Pascal style real to double.

Declaration: `function Real2Double(r: Real48) : Double`

Visibility: default

Description: The `Real2Double` function converts a Turbo Pascal style real (6 bytes long) to a native Free Pascal double type. It can be used e.g. to read old binary TP files with FPC and convert them to Free Pascal binary files.

Note that the assignment operator has been overloaded so a `Real48` type can be assigned directly to a double or extended.

Errors: None.

Listing: `./examples/refex/ex110.pp`

```

program Example110;

{ Program to demonstrate the Real2Double function. }

Var
  i : integer;
  R : Real48;
  D : Double;
  E : Extended;
  F : File of Real48;

begin
  Assign(F,'reals.dat');
  Reset(f);
  For i:=1 to 10 do
    begin
      Read(F,R);
      D:=Real2Double(R);
      Writeln('Real ',i,' : ',D);
      D:=R;
      Writeln('Real (direct to double) ',i,' : ',D);
      E:=R;
      Writeln('Real (direct to Extended) ',i,' : ',E);
    end;
  Close(f);
end.

```

75.12.260 ReAllocMem

Synopsis: Re-allocate memory on the heap.

Declaration: `function ReAllocMem(var p: pointer; Size: PtrUInt) : pointer`

Visibility: default

Description: `ReAllocMem` resizes the memory pointed to by `P` so it has size `Size`. The value of `P` may change during this operation. The contents of the memory pointed to by `P` (if any) will be copied to the new location, but may be truncated if the newly allocated memory block is smaller in size. If a larger block is allocated, only the used memory is initialized, extra memory will not be zeroed out.

Note that `P` may be `nil`, in that case the behaviour of `ReAllocMem` is equivalent to `Getmem`.

See also: `GetMem` ([1474](#)), `FreeMem` ([1471](#))

75.12.261 ReAllocMemory

Synopsis: Alias for `ReAllocMem` ([1523](#)).

Declaration: `function ReAllocMemory(p: pointer; Size: PtrUInt) : pointer`

Visibility: default

Description: `ReAllocMemory` is an alias for `ReAllocMem` ([1523](#)).

See also: `ReAllocMem` ([1523](#))

75.12.262 RegisterLazyInitThreadingProc

Declaration: `procedure RegisterLazyInitThreadingProc(const proc: TProcedure)`

Visibility: default

75.12.263 ReleaseExceptionObject

Synopsis: Decrease the reference count of the current exception object.

Declaration: `procedure ReleaseExceptionObject`

Visibility: default

Description: `ReleaseExceptionObject` decreases the reference count of the current exception object. This should be called whenever a reference to the exception object was obtained via the `AcquireExceptionObject` ([1419](#)) call.

Calling this method is only valid within an `except` block.

Errors: If there is no current exception object, a run-time error 231 will occur.

See also: `AcquireExceptionObject` ([1419](#))

75.12.264 Rename

Synopsis: Rename file on disk.

Declaration: `procedure Rename (var f: File; const s: ShortString)`
`procedure Rename (var f: File; const p: PAnsiChar)`
`procedure Rename (var f: File; const c: AnsiChar)`
`procedure Rename (var f: File; const s: UnicodeString)`
`procedure Rename (var f: File; const s: RawByteString)`
`procedure Rename (var t: Text; const s: shortstring)`
`procedure Rename (var t: Text; const p: PAnsiChar)`
`procedure Rename (var t: Text; const c: AnsiChar)`
`procedure Rename (var t: Text; const s: unicodestring)`
`procedure Rename (var t: Text; const s: RawByteString)`

Visibility: default

Description: `Rename` changes the name of the assigned file `F` to `S`. `F` must be assigned, but not opened.

Errors: Depending on the state of the `{SI}` switch, a runtime error can be generated if there is an error. In the `{SI-}` state, use `IOResult` to check for errors.

See also: `Erase` ([1458](#))

Listing: `./examples/refex/ex77.pp`

Program `Example77;`

{ Program to demonstrate the Rename function. }

Var `F : Text;`

begin

`Assign (F,paramstr(1));`

`Rename (F,paramstr(2));`

end.

75.12.265 Reset

Synopsis: Open file for reading.

Declaration: `procedure Reset (var f: File; l: LongInt)`
`procedure Reset (var f: File)`
`procedure Reset (var f: TypedFile)`
`procedure Reset (var t: Text)`

Visibility: default

Description: `Reset` opens a file `F` for reading. `F` can be any file type. If `F` is a text file, or refers to standard I/O (e.g. `”) then it is opened read-only, otherwise it is opened using the mode specified in filemode` ([1345](#)).

If `F` is an untyped file, the record size can be specified in the optional parameter `L`. A default value of 128 is used.

File sharing is not taken into account when calling `Reset`.

Note that the path can be only 255 characters long.

Errors: Depending on the state of the `{SI}` switch, a runtime error can be generated if there is an error. In the `{SI-}` state, use `IOResult` to check for errors.

See also: Rewrite ([1525](#)), Assign ([1423](#)), Close ([1437](#)), Append ([1421](#)), FileMode ([1345](#))

Listing: ./examples/refex/ex51.pp

Program Example51;

{ Program to demonstrate the Reset function. }

Function FileExists (Name : String) : boolean;

Var F : File;

begin

{ \$i- }

Assign (F, Name);

Reset (F);

{ \$i+ }

FileExists := (IoResult = 0) and (Name <> "");

if FileExists **then**

Close (f);

end;

begin

If FileExists (Paramstr(1)) **then**

Writeln ('File found')

else

Writeln ('File NOT found');

end.

75.12.266 ResumeThread

Synopsis: Resume a suspended thread.

Declaration: `function ResumeThread(threadHandle: TThreadID) : DWord`

Visibility: default

Description: ResumeThread causes a suspended thread (using SuspendThread ([1555](#))) to resume it's execution.

The thread is identified with it's handle or ID threadHandle. This function is only usable on windows, it will return an error value on linux.

The function returns a positive number if successful. A DWord (-1) return value indicates failure.

Errors: If a failure occurred, a DWord (-1) result is returned.

See also: SuspendThread ([1555](#)), KillThread ([1499](#))

75.12.267 Rewrite

Synopsis: Open file for writing.

Declaration: `procedure Rewrite(var f: File; l: LongInt)`

`procedure Rewrite(var f: File)`

`procedure Rewrite(var f: TypedFile)`

`procedure Rewrite(var t: Text)`

Visibility: default

Description: `Rewrite` opens a file `F` for writing. `F` can be any file type. If `F` is an untyped or typed file, then it is opened for reading and writing. If `F` is an untyped file, the record size can be specified in the optional parameter `L`. Default a value of 128 is used. if `Rewrite` finds a file with the same name as `F`, this file is truncated to length 0. If it doesn't find such a file, a new file is created.

Contrary to Turbo Pascal, Free Pascal opens the file with mode `fmoutput`. If it should be opened in `fminout` mode, an extra call to `Reset` (1524) is needed.

File sharing is not taken into account when calling `Rewrite`.

Note that the path can be only 255 characters long.

Errors: Depending on the state of the `{SI}` switch, a runtime error can be generated if there is an error. In the `{SI-}` state, use `IOResult` to check for errors.

See also: `Reset` (1524), `Assign` (1423), `Close` (1437), `Flush` (1469), `Append` (1421)

Listing: `./examples/refex/ex52.pp`

Program Example52;

{ Program to demonstrate the Rewrite function. }

Var `F : File;`
 `I : longint;`

begin

`Assign (F,'Test.tmp');`
 { Create the file. Recordsize is 4 }
 `Rewrite (F,Sizeof(I));`
 For `I:=1 to 10 do`
 `BlockWrite (F,I,1);`
 `close (f);`
 { F contains now a binary representation of
 10 longints going from 1 to 10 }

end.

75.12.268 `rightshift(variant,variant):variant`

Synopsis: Implement binary `shr` operation on variants.

Declaration: `operator shr(const op1: variant; const op2: variant) : variant`

Visibility: default

Description: The implementation of the `shr` operation is delegated to the variant manager with operation `opshiftright`.

Errors: Execution of this operator may result in an exception if no variant manager is installed or if the types of the operand are not suitable for the operation.

See also: `operator shl(variant, variant): variant` (1335)

75.12.269 `RmDir`

Synopsis: Remove directory when empty.

Declaration: `procedure RmDir(const s: shortstring); Overload`
 `procedure RmDir(const s: RawByteString); Overload`
 `procedure RmDir(const s: unicodestring); Overload`

Visibility: default

Description: `Rmdir` removes the directory `S`.

Errors: Depending on the state of the `{SI}` switch, a runtime error can be generated if there is an error. In the `{SI-}` state, use `IOResult` to check for errors.

See also: `Chdir` ([1436](#)), `Mkdir` ([1507](#))

Listing: `./examples/refex/ex53.pp`

Program Example53;

{ Program to demonstrate the Mkdir and Rmdir functions. }

Const `D : String[8]` = 'TEST.DIR';

Var `S : String`;

begin

Writeln ('Making directory ',D);

Mkdir (D);

Writeln ('Changing directory to ',D);

ChDir (D);

GetDir (0,S);

Writeln ('Current Directory is : ',S);

WRiteln ('Going back');

ChDir ('..');

Writeln ('Removing directory ',D);

Rmdir (D);

end.

75.12.270 RolByte

Synopsis: Rotate bits of a byte value to the left.

Declaration: `function RolByte(const AValue: Byte) : Byte`
 `function RolByte(const AValue: Byte; const Dist: Byte) : Byte`

Visibility: default

Description: `RolByte` rotates the bits of the byte `AValue` with `Dist` positions to the left. If `Dist` is not specified, then 1 is assumed.

Errors: None.

See also: `RorByte` ([1528](#)), `RolWord` ([1528](#)), `RolDWord` ([1527](#)), `RolQWord` ([1528](#))

75.12.271 RolDWord

Synopsis: Rotate bits of a DWord (cardinal) value to the left.

Declaration: `function RolDWord(const AValue: DWord) : DWord`
 `function RolDWord(const AValue: DWord; const Dist: Byte) : DWord`

Visibility: default

Description: `RolDWord` rotates the bits of the `DWord` (cardinal) `AValue` with `Dist` positions to the left. If `Dist` is not specified, then 1 is assumed.

Errors: None.

See also: `RolByte` ([1527](#)), `RolWord` ([1528](#)), `RorDWord` ([1529](#)), `RolQWord` ([1528](#))

75.12.272 `RolQWord`

Synopsis: Rotate bits of a `QWord` (64-bit) value to the left.

Declaration: `function RolQWord(const AValue: QWord) : QWord`
`function RolQWord(const AValue: QWord; const Dist: Byte) : QWord`

Visibility: default

Description: `RorQWord` rotates the bits of the `QWord` (64-bit) `AValue` with `Dist` positions to the left. If `Dist` is not specified, then 1 is assumed.

Errors: None.

See also: `RolByte` ([1527](#)), `RolWord` ([1528](#)), `RolDWord` ([1527](#)), `RorQWord` ([1529](#))

75.12.273 `RolWord`

Synopsis: Rotate bits of a word value to the left.

Declaration: `function RolWord(const AValue: Word) : Word`
`function RolWord(const AValue: Word; const Dist: Byte) : Word`

Visibility: default

Description: `RolWord` rotates the bits of the word `AValue` with `Dist` positions to the right. If `Dist` is not specified, then 1 is assumed.

Errors: None.

See also: `RolByte` ([1527](#)), `RorWord` ([1529](#)), `RolDWord` ([1527](#)), `RolQWord` ([1528](#))

75.12.274 `RorByte`

Synopsis: Rotate bits of a byte value to the right.

Declaration: `function RorByte(const AValue: Byte) : Byte`
`function RorByte(const AValue: Byte; const Dist: Byte) : Byte`

Visibility: default

Description: `RorByte` rotates the bits of the byte `AValue` with `Dist` positions to the right. If `Dist` is not specified, then 1 is assumed.

Errors: None.

See also: `RolByte` ([1527](#)), `RorWord` ([1529](#)), `RorDWord` ([1529](#)), `RorQWord` ([1529](#))

75.12.275 RorDWord

Synopsis: Rotate bits of a DWord (cardinal) value to the right.

Declaration: `function RorDWord(const AValue: DWord) : DWord`
`function RorDWord(const AValue: DWord; const Dist: Byte) : DWord`

Visibility: default

Description: `RorDWord` rotates the bits of the DWord (cardinal) `AValue` with `Dist` positions to the right. If `Dist` is not specified, then 1 is assumed.

Errors: None.

See also: `RorByte` ([1528](#)), `RolDWord` ([1527](#)), `RorWord` ([1529](#)), `RorQWord` ([1529](#))

75.12.276 RorQWord

Synopsis: Rotate bits of a QWord (64-bit) value to the right.

Declaration: `function RorQWord(const AValue: QWord) : QWord`
`function RorQWord(const AValue: QWord; const Dist: Byte) : QWord`

Visibility: default

Description: `RorQWord` rotates the bits of the QWord (64-bit) `AValue` with `Dist` positions to the right. If `Dist` is not specified, then 1 is assumed.

Errors: None.

See also: `RorByte` ([1528](#)), `RorWord` ([1529](#)), `RorDWord` ([1529](#)), `RolQWord` ([1528](#))

75.12.277 RorWord

Synopsis: Rotate bits of a word value to the right.

Declaration: `function RorWord(const AValue: Word) : Word`
`function RorWord(const AValue: Word; const Dist: Byte) : Word`

Visibility: default

Description: `RorWord` rotates the bits of the word `AValue` with `Dist` positions to the right. If `Dist` is not specified, then 1 is assumed.

Errors: None.

See also: `RorByte` ([1528](#)), `RolWord` ([1528](#)), `RorDWord` ([1529](#)), `RorQWord` ([1529](#))

75.12.278 Round

Synopsis: Round floating point value to nearest integer number.

Declaration: `function Round(d: ValReal) : Int64`

Visibility: default

Description: `Round` rounds `X` to the closest integer, which may be bigger or smaller than `X`.

In the case of .5, the algorithm uses "banker's rounding": .5 values are always rounded towards the even number.

Errors: An invalid floating point operation error may occur in one of several conditions:

- Infinity or NegInfinity is specified.
- Nan is specified.
- X is out of the valid range of integers.

See also: [Frac \(1471\)](#), [Int \(1493\)](#), [Trunc \(1561\)](#)

Listing: ./examples/refex/ex54.pp

Program Example54;

{ Program to demonstrate the Round function. }

```
begin
  Writeln (Round(1234.56)); { Prints 1235 }
  Writeln (Round(-1234.56)); { Prints -1235 }
  Writeln (Round(12.3456)); { Prints 12 }
  Writeln (Round(-12.3456)); { Prints -12 }
  Writeln (Round(2.5)); { Prints 2 (down) }
  Writeln (Round(3.5)); { Prints 4 (up) }
```

end.

Listing: ./examples/refex/ex125.pp

program ex124;

```
{
  Example to show various error conditions for the Round() function.
  SysUtils is used to convert the runtime error to an exception that can be caught
  Math is used to have access to Nan and (Neg)Infinity
}
```

```
{ $mode objfpc }
```

```
{ $h+ }
```

uses

```
  sysutils, math;
```

var

```
  D: Double;
```

```
  Q: QWord;
```

```
  L: Int64;
```

begin

```
  D := 2.0 * High(QWord);
```

```
  try
```

```
    Q := Round(D);
```

```
  except
```

```
    on E: Exception do writeln(E.ClassName, ': ', E.Message); //EInvalidOp: Invalid floating point operation
```

```
  end;
```

```
  D := NaN;
```

```
  try
```

```
    Q := Round(D);
```

```
  except
```

```
    on E: Exception do writeln(E.ClassName, ': ', E.Message); //EInvalidOp: Invalid floating point operation
```

```
  end;
```

```
  D := Infinity;
```

```

try
  Q := Round(D);
except
  on E: Exception do writeln(E.ClassName,': ',E.Message); //EInvalidOp: Invalid floating point operation
end;
D := NegInfinity;
try
  Q := Round(D);
except
  on E: Exception do writeln(E.ClassName,': ',E.Message); //EInvalidOp: Invalid floating point operation
end;
D := -2.0 * High(QWord);
try
  L := Round(D);
except
  on E: Exception do writeln(E.ClassName,': ',E.Message); //EInvalidOp: Invalid floating point operation
end;
end.

```

75.12.279 RTLEventCreate

Synopsis: Create a new RTL event.

Declaration: `function RTLEventCreate : PRTLEvent`

Visibility: default

Description: `RTLEventCreate` creates and initializes a new RTL event. RTL events are used to notify other threads that a certain condition is met, and to notify other threads of condition changes (conditional variables).

The function returns an initialized RTL event, which must be disposed of with `RTLEventdestroy` ([1531](#))

`RTLEvent` is used mainly for the `synchronize` method.

See also: `RTLEventDestroy` ([1531](#)), `RTLEventSetEvent` ([1532](#)), `RTLEventReSetEvent` ([1531](#)), `RTLEventWaitFor` ([1532](#))

75.12.280 RTLEventDestroy

Synopsis: Destroy a RTL Event.

Declaration: `procedure RTLEventDestroy(state: PRTLEvent)`

Visibility: default

Description: `RTLeventdestroy` destroys the RTL event `State`. After a call to `RTLeventdestroy`, the `State` RTL event may no longer be used.

See also: `RTLEventCreate` ([1531](#)), `RTLEventResetEvent` ([1531](#)), `RTLEventSetEvent` ([1532](#))

75.12.281 RTLEventResetEvent

Synopsis: Reset an event.

Declaration: `procedure RTLEventResetEvent(state: PRTLEvent)`

Visibility: default

Description: `RTLeventResetEvent` resets the event: this should be used to undo the signaled state of an event. Resetting an event that is not set (or was already reset) has no effect.

See also: `RTLeventCreate` ([1531](#)), `RTLeventDestroy` ([1531](#)), `RTLeventSetEvent` ([1532](#)), `RTLeventWaitFor` ([1532](#))

75.12.282 RTLeventSetEvent

Synopsis: Notify threads of the event.

Declaration: `procedure RTLeventSetEvent (state: PRTLevent)`

Visibility: default

Description: `RTLeventSetEvent` notifies other threads which are listening, that the event has occurred.

See also: `RTLeventCreate` ([1531](#)), `RTLeventResetEvent` ([1531](#)), `RTLeventDestroy` ([1531](#)), `RTLeventWaitFor` ([1532](#))

75.12.283 RTLeventWaitFor

Synopsis: Wait for an event.

Declaration: `procedure RTLeventWaitFor (state: PRTLevent)`
`procedure RTLeventWaitFor (state: PRTLevent; timeout: LongInt)`

Visibility: default

Description: `RTLeventWaitFor` suspends the thread till the event occurs. The event will occur when another thread calls `RTLeventSetEvent` ([1532](#)) on `State`.

By default, the thread will be suspended indefinitely. However, if `TimeOut` is specified, then the thread will resume after timeout milliseconds have elapsed.

See also: `RTLeventCreate` ([1531](#)), `RTLeventDestroy` ([1531](#)), `RTLeventSetEvent` ([1532](#)), `RTLeventWaitFor` ([1532](#))

75.12.284 RunError

Synopsis: Generate a run-time error.

Declaration: `procedure RunError (w: Word)`
`procedure RunError`

Visibility: default

Description: `Runerror` stops the execution of the program, and generates a run-time error `ErrorCode`.

Errors: None.

See also: `Exit` ([1460](#)), `Halt` ([1481](#))

Listing: `./examples/refex/ex55.pp`

Program Example55;

{ Program to demonstrate the RunError function. }

begin
{ The program will stop and emit a run-error 106 }
RunError (106);
end.

75.12.285 SafeLoadLibrary

Synopsis: Load a library safely.

Declaration: `function SafeLoadLibrary(const Name: RawByteString) : TLibHandle`
`function SafeLoadLibrary(const Name: UnicodeString) : TLibHandle`

Visibility: default

Description: `SafeLoadLibrary` calls `LoadLibrary` ([1503](#)) but restores the current FPU control word and exception mask to their current value after the library was loaded, thus preventing the loaded library initialization code from modifying their current values.

See also: `LoadLibrary` ([1503](#))

75.12.286 SarInt64

Synopsis: 64-bit Shift Arithmetic Right.

Declaration: `function SarInt64(const AValue: Int64) : Int64`
`function SarInt64(const AValue: Int64; Shift: Byte) : Int64`

Visibility: default

Description: `SarInt64` performs an arithmetic right shift for `Shift` positions on a 64-bit integer `AValue` and returns the result. `Shift` is optional, and is 1 by default. The difference with the regular `Shr` shift operation is that the leftmost bit is preserved during the shift operation.

See also: `SarShortInt` ([1534](#)), `SarSmallInt` ([1534](#)), `SarLongInt` ([1533](#))

75.12.287 SarLongint

Synopsis: 32-bit Shift Arithmetic Right.

Declaration: `function SarLongint(const AValue: LongInt; const Shift: Byte) : LongInt`

Visibility: default

Description: `SarLongint` performs an arithmetic right shift for `Shift` positions on a 32-bit integer `AValue` and returns the result. `Shift` is optional, and is 1 by default. The difference with the regular `Shr` shift operation is that the leftmost bit is preserved during the shift operation.

See also: `SarShortInt` ([1534](#)), `SarSmallInt` ([1534](#)), `SarInt64` ([1533](#))

75.12.288 SarShortint

Synopsis: 8-bit Shift Arithmetic Right.

Declaration: `function SarShortint(const AValue: ShortInt; const Shift: Byte)
: ShortInt`

Visibility: default

Description: `SarShortint` performs an arithmetic right shift for `Shift` positions on an 8-bit integer `AValue` and returns the result. `Shift` is optional, and is 1 by default. The difference with the regular `Shr` shift operation is that the leftmost bit is preserved during the shift operation.

See also: `SarSmallint` ([1534](#)), `SarLongint` ([1533](#)), `SarInt64` ([1533](#))

75.12.289 SarSmallint

Synopsis: 16-bit Shift Arithmetic Right.

Declaration: `function SarSmallint(const AValue: SmallInt; const Shift: Byte)
: SmallInt`

Visibility: default

Description: `SarSmallint` performs an arithmetic right shift for `Shift` positions on an 16-bit integer `AValue` and returns the result. `Shift` is optional, and is 1 by default. The difference with the regular `Shr` shift operation is that the leftmost bit is preserved during the shift operation.

See also: `SarShortint` ([1534](#)), `SarLongint` ([1533](#)), `SarInt64` ([1533](#))

75.12.290 Seek

Synopsis: Set file position.

Declaration: `procedure Seek(var f: File; Pos: Int64)`

Visibility: default

Description: `Seek` sets the file-pointer for file `F` to record `Nr. Count`. The first record in a file has `Count=0`. `F` can be any file type, except `Text`. If `F` is an untyped file, with no record size specified in `Reset` ([1524](#)) or `Rewrite` ([1525](#)), 128 is assumed.

Errors: Depending on the state of the `{SI}` switch, a runtime error can be generated if there is an error. In the `{SI-}` state, use `IOResult` to check for errors.

See also: `Eof` ([1456](#)), `SeekEof` ([1535](#)), `SeekEoln` ([1536](#))

Listing: `./examples/refex/ex56.pp`

Program Example56;

{ Program to demonstrate the Seek function. }

Var

`F : File;`

`l,j : longint;`

begin

{ Create a file and fill it with data }

```

Assign (F,'test.tmp');
Rewrite(F); { Create file }
Close(f);
FileMode:=2;
ReSet (F,Sizeof(i)); { Opened read/write }
For I:=0 to 10 do
  BlockWrite (F,I,1);
{ Go Back to the begining of the file }
Seek(F,0);
For I:=0 to 10 do
  begin
    BlockRead (F,J,1);
    If J<>I then
      Writeln ('Error: expected ',i,', got ',j);
    end;
  end;
Close (f);
end.

```

75.12.291 SeekEOF

Synopsis: Set file position to end of file.

Declaration: `function SeekEOF (var t: Text) : Boolean`
`function SeekEOF : Boolean`

Visibility: default

Description: `SeekEof` returns `True` is the file-pointer is at the end of the file. It ignores all whitespace. Calling this function has the effect that the file-position is advanced until the first non-whitespace character or the end-of-file marker is reached.

If the end-of-file marker is reached, `True` is returned. Otherwise, `False` is returned.

If the parameter `F` is omitted, standard `Input` is assumed.

Remark The `SeekEOF` function can only be used on real textfiles: when assigning the file to other kinds of (virtual) text files, the function may fail, although it will perform a number of tests to guard against wrong usage.

Errors: A run-time error is generated if the file `F` isn't opened.

See also: `Eof` ([1456](#)), `SeekEoln` ([1536](#)), `Seek` ([1534](#))

Listing: `./examples/refex/ex57.pp`

Program Example57;

```

{ Program to demonstrate the SeekEof function. }
Var C : Char;

begin
  { this will print all characters from standard input except
    Whitespace characters. }
  While Not SeekEof do
    begin
      Read (C);
      Write (C);
    end;
end.

```

75.12.292 SeekEOLn

Synopsis: Set file position to end of line.

Declaration: `function SeekEOLn(var t: Text) : Boolean`
`function SeekEOLn : Boolean`

Visibility: default

Description: `SeekEOLn` returns `True` if the file-pointer is at the end of the current line. It ignores all whitespace. Calling this function has the effect that the file-position is advanced until the first non-whitespace character or the end-of-line marker is reached. If the end-of-line marker is reached, `True` is returned. Otherwise, `False` is returned. The end-of-line marker is defined as `#10`, the LineFeed character. If the parameter `F` is omitted, standard `Input` is assumed.

Errors: A run-time error is generated if the file `F` isn't opened.

See also: `Eof` ([1456](#)), `SeekEof` ([1535](#)), `Seek` ([1534](#))

Listing: `./examples/refex/ex58.pp`

Program Example58;

```
{ Program to demonstrate the SeekEOLn function. }
Var
  C : Char;

begin
  { This will read the first line of standard output and print
    all characters except whitespace. }
  While not SeekEOLn do
    Begin
      Read (c);
      Write (c);
    end;
end.
```

75.12.293 Seg

Synopsis: Return segment.

Declaration: `function Seg(var X) : LongInt`

Visibility: default

Description: `Seg` returns the segment of the address of a variable. This function is only supported for compatibility. In Free Pascal, it returns always 0, since Free Pascal uses a flat 32/64 bit memory model. In such a memory model segments have no meaning.

Errors: None.

See also: `DSeg` ([1452](#)), `CSeg` ([1447](#)), `Ofs` ([1512](#)), `Ptr` ([1517](#))

Listing: `./examples/refex/ex60.pp`

Program Example60;

```
{ Program to demonstrate the Seg function. }
Var
```

W : Word;

begin

W:=Seg(W); { W contains its own Segment}

end.

75.12.294 Set8087CW

Declaration: procedure Set8087CW(cw: Word)

Visibility: default

75.12.295 SetCodePage

Synopsis: Set the codepage of a string.

Declaration: procedure SetCodePage(var s: RawByteString; CodePage: TSystemCodePage;
Convert: Boolean)

Visibility: default

Description: SetCodePage sets the codepage of a string S to CodePage. If Convert is True then the string will be transcoded to the new codepage. The resulting string will have reference count 1.

See also: StringCodePage ([1551](#))

75.12.296 SetDynLibsManager

Synopsis: Set a new handler for dynamic library support.

Declaration: procedure SetDynLibsManager(const &New: TDynLibsManager)
procedure SetDynLibsManager(const &New: TDynLibsManager;
var Old: TDynLibsManager)

Visibility: default

Description: SetDynLibsManager sets the handler for dynamic library support to New. Optionally, it returns the currently active handler in Old.

This is the routine used by the #rtl.dynlibs ([754](#)) unit to set the default handler.

See also: #rtl.dynlibs ([754](#)), GetDynLibsManager ([1474](#))

75.12.297 Setjmp

Synopsis: Save current execution point.

Declaration: function Setjmp(var S: jmp_buf) : LongInt

Visibility: default

Description: SetJmp fills S with the necessary data for a jump back to the point where it was called. It returns zero if called in this way. If the function returns nonzero, then it means that a call to LongJmp ([1505](#)) with S as an argument was made somewhere in the program.

Errors: None.

See also: LongJump ([1505](#))

Listing: ./examples/refex/ex79.pp

```

program example79;

{ Program to demonstrate the setjmp, longjmp functions }

procedure dojmp(var env : jmp_buf; value : longint);

begin
  value:=2;
  Writeln ('Going to jump !');
  { This will return to the setjmp call,
    and return value instead of 0 }
  longjmp(env,value);
end;

var env : jmp_buf;

begin
  if setjmp(env)=0 then
    begin
      writeln ('Passed first time.');
```

75.12.298 SetLength

Synopsis: Set length of a string or dynamic array.

Declaration: `procedure &SetLength(var S: AStringType; Len: SizeInt)`
`procedure &SetLength(var A: DynArrayType; Len: SizeInt)`

Visibility: default

Description: SetLength for strings sets the length of the string S to Len. S can be an ansistring, a short string or a widestring. For ShortStrings, Len can maximally be 255. For AnsiStrings it can have any value. For AnsiString strings, SetLength **must** be used to set the length of the string.

In the case of a dynamic array A, SetLength sets the number of elements. The elements are numbered from index 0, so the count runs from 0 to Len-1. If Zero is specified, the array is cleared and removed from memory.

In case the length is set to a smaller length than the current one, the elements 0-(Len-1) (or characters 1-Len in case of a string) are kept. The elements that fall outside the new length are finalized if the array element is a managed type.

In case the length is set to a larger length than the current one, the new elements are zeroed out for a dynamic array. For a string, the string is zero-terminated at the correct length.

Note that SetLength is governed by the Copy-On-Write principle for strings and dynamic arrays: the reference count after a call to SetLength will be 1 (except when the length is zero, then the array is removed from memory).

For multi-dimensional arrays, SetLength can be used to specify all dimensions at once:

```

Var
  arr : Array of Array of Integer;

begin
  SetLength(arr, 10, 20);
end;

```

This will create a dynamic array with 10 elements, where each element in itself is a dynamic array of 20 elements. `SetLength` will of course always create "rectangular" arrays: all elements will have the same size.

```

Var
  arr : Array of Integer;

begin
  SetLength(arr, 0);
end;

```

After this, `arr` is `Nil`.

Errors: None.

See also: [Length \(1500\)](#)

Listing: `./examples/refex/ex85.pp`

Program Example85;

{ Program to demonstrate the SetLength function. }

Var S : String;

```

begin
  Setlength(S,100);
  FillChar(S[1],100,#32);
  Writeln ("",S,"");
end.

```

75.12.299 SetMemoryManager

Synopsis: Set a memory manager.

Declaration: `procedure SetMemoryManager(const MemMgr: TMemoryManager)`

Visibility: default

Description: `SetMemoryManager` sets the current memory manager record to `MemMgr`.

For an example, see the programmer's guide.

Errors: None.

See also: [GetMemoryManager \(1475\)](#), [IsMemoryManagerSet \(1499\)](#)

75.12.300 SetMultiByteConversionCodePage

Synopsis: Set codepage for conversions from multi-byte strings to single-byte strings.

Declaration: `procedure SetMultiByteConversionCodePage (CodePage: TSystemCodePage)`

Visibility: default

Description: `SetMultiByteConversionCodePage` sets `DefaultSystemCodePage` (1415) to `CodePage`. The effect of this change is that the default codepage used to translate multi-byte (UTF-16) strings to single-byte codepage-aware strings changes, and code page conversions will be done to the new codepage.

Do not set `DefaultSystemCodePage` directly, as additional actions may need to be done when changing the code page.

When the `DefaultSystemCodePage` (1415) changes and you use the `TEncoding` classes, you must call `TEncoding.FreeEncodings` (1871) to regenerate the default encoding using the new code page.

See also: `DefaultSystemCodePage` (1415), `SetMultiByteFileSystemCodePage` (1540), `SetMultiByteRTLFileSystemCodePage` (1540)

75.12.301 SetMultiByteFileSystemCodePage

Synopsis: Set codepage used when passing strings to OS single-byte file system APIs.

Declaration: `procedure SetMultiByteFileSystemCodePage (CodePage: TSystemCodePage)`

Visibility: default

Description: `SetMultiByteFileSystemCodePage` sets the codepage used in single-byte OS file system APIs to `CodePage`. The effect of this change is that the default codepage used to translate multi-byte (UTF-16) strings to single-byte codepage-aware strings used in File system APIs changes, and strings passed to the codepage-aware file system APIs will be passed using the new codepage.

This constant is not used if the file system API of the OS is multi-byte (such as on Windows).

Do not set `DefaultFileSystemCodePage` directly, as additional actions may need to be done when changing the code page.

See also: `DefaultFileSystemCodePage` (1414), `SetMultiByteConversionCodePage` (1540), `SetMultiByteRTLFileSystemCodePage` (1540)

75.12.302 SetMultiByteRTLFileSystemCodePage

Synopsis: Set codepage used when interpreting strings from OS single-byte file system APIs.

Declaration: `procedure SetMultiByteRTLFileSystemCodePage (CodePage: TSystemCodePage)`

Visibility: default

Description: `SetMultiByteRTLFileSystemCodePage` sets the codepage used to interpret strings returned by single-byte OS file system APIs to `CodePage`.

The effect of this change is that the default codepage used to translate single byte strings obtained from the OS to single-byte codepage-aware strings or multi-byte strings changes, and strings obtained from the codepage-aware file system APIs will be interpreted using the new codepage.

This constant is not used if the file system API of the OS is multi-byte (such as on Windows).

Do not set `DefaultRTLFileSystemCodePage` directly, as additional actions may need to be done when changing the code page.

See also: `SetMultiByteFileSystemCodePage` ([1540](#)), `SetMultiByteConversionCodePage` ([1540](#)), `SetMultiByteRTL-FileSystemCodePage` ([1540](#))

75.12.303 SetMXCSR

Declaration: `procedure SetMXCSR(w: DWord)`

Visibility: default

75.12.304 SetResourceManager

Synopsis: Set the resource manager.

Declaration: `procedure SetResourceManager(const &New: TResourceManager)`

Visibility: default

Description: `SetResourceManager` sets the active resource manager to `Manager`. After a call to `SetResourceManager`, the functions in the `Manager` record will be used to handle resources.

Note that it is not supported to change resource managers on-the-fly: any resources or information about resources obtained should be discarded prior to a call to `SetResourceManager`. Typically, `SetResourceManager` should be called once, at program startup.

Errors: None.

See also: `TResourceManager` ([1396](#)), `GetResourceManager` ([1476](#))

75.12.305 SetSSECSR

Declaration: `procedure SetSSECSR(w: DWord)`

Visibility: default

75.12.306 SetString

Synopsis: Set length of a string and copy buffer.

Declaration: `procedure SetString(out S: AnsiString; Buf: PAnsiChar; Len: SizeInt)`
`procedure SetString(out S: AnsiString; Buf: PWideChar; Len: SizeInt)`
`procedure SetString(out S: Shortstring; Buf: PChar; Len: SizeInt)`
`procedure SetString(out S: UnicodeString; Buf: PUnicodeChar;`
`Len: SizeInt)`
`procedure SetString(out S: UnicodeString; Buf: PChar; Len: SizeInt)`
`procedure SetString(out S: WideString; Buf: PWideChar; Len: SizeInt)`
`procedure SetString(out S: WideString; Buf: PChar; Len: SizeInt)`

Visibility: default

Description: `SetString` sets the length of the string `S` to `Len` and if `Buf` is non-nil, copies `Len` characters from `Buf` into `S`. `S` can be an ansistring, a short string or a widestring. For `ShortStrings`, `Len` can maximally be 255.

Errors: None.

See also: `SetLength` ([1538](#))

75.12.307 SetTextAutoFlush

Declaration: `procedure SetTextAutoFlush(var T: Text; AutoFlush: Boolean)`

Visibility: default

75.12.308 SetTextBuf

Synopsis: Set size of text file internal buffer.

Declaration: `procedure SetTextBuf(var f: Text; var Buf)
procedure SetTextBuf(var f: Text; var Buf; Size: SizeInt)`

Visibility: default

Description: `SetTextBuf` assigns an I/O buffer to a text file. The new buffer is located at `Buf` and is `Size` bytes long. If `Size` is omitted, then `SizeOf (Buf)` is assumed. The standard buffer of any text file is 128 bytes long. For heavy I/O operations this may prove too slow. The `SetTextBuf` procedure allows to set a bigger buffer for the I/O of the application, thus reducing the number of system calls, and thus reducing the load on the system resources. The maximum size of the newly assigned buffer is 65355 bytes.

Remark

- Never assign a new buffer to an opened file. A new buffer can be assigned immediately after a call to `Rewrite` (1525), `Reset` (1524) or `Append`, but not after the file was read from/written to. This may cause loss of data. If a new buffer must be assigned after read/write operations have been performed, the file should be flushed first. This will ensure that the current buffer is emptied.
- Take care that the assigned buffer is always valid. If a local variable is assigned as a buffer, then after the program exits the local program block, the buffer will no longer be valid, and stack problems may occur.

Errors: No checking on `Size` is done.

See also: `Assign` (1423), `Reset` (1524), `Rewrite` (1525), `Append` (1421)

Listing: `./examples/refex/ex61.pp`

Program Example61;

{ Program to demonstrate the SetTextBuf function. }

Var

`Fin, Fout : Text;
Ch : Char;
Bufin, Bufout : Array[1..10000] of byte;`

begin

`Assign (Fin,paramstr(1));
Reset (Fin);
Assign (Fout,paramstr(2));
Rewrite (Fout);
{ This is harmless before IO has begun }
{ Try this program again on a big file,
after commenting out the following 2
lines and recompiling it. }
SetTextBuf (Fin,Bufin);`

```

SetTextBuf (Fout,Bufout);
While not eof(Fin) do
  begin
    Read (Fin,ch);
    write (Fout,ch);
  end;
  Close (Fin);
  Close (Fout);
end.

```

75.12.309 SetTextCodePage

Synopsis: Set the codepage used in a text file.

Declaration: `procedure SetTextCodePage (var T: Text; CodePage: TSystemCodePage)`

Visibility: default

Description: `SetTextCodePage` sets the codepage that the text file `T` uses. All strings written to the file will be converted to the indicated codepage. By default, the codepage is set to `CP_ACP`.

Errors: None.

See also: `TextRec` ([1389](#)), `GetTextCodePage` ([1477](#))

75.12.310 SetTextLineEnding

Synopsis: Set the end-of-line character for the given text file.

Declaration: `procedure SetTextLineEnding (var f: Text; Ending: string)`

Visibility: default

Description: `SetTextLineEnding` sets the end-of-line character for the text file `F` to `Ending`. By default, this is the string indicated by `DefaultTextLineBreakStyle` ([1344](#)).

Errors: None.

See also: `DefaultTextLineBreakStyle` ([1344](#)), `TTextLineBreakStyle` ([1399](#))

75.12.311 SetThreadDebugName

Synopsis: Set the debug name for a thread.

Declaration: `procedure SetThreadDebugName (threadHandle: TThreadID;
 const ThreadName: AnsiString)
 procedure SetThreadDebugName (threadHandle: TThreadID;
 const ThreadName: UnicodeString)`

Visibility: default

Description: `SetThreadDebugName` can be used to set the name of a thread in a debugging context. A debugger can use this information to show the name instead of the thread ID, thus making debugging easier. The current platform's `TThreadManager` ([1400](#)) implementation must support this call.

Errors: This call does nothing on platforms that do not support threading.

See also: `GetCurrentThreadID` ([1473](#)), `TThreadManager` ([1400](#))

75.12.312 SetThreadManager

Synopsis: Set the thread manager, optionally return the current thread manager.

Declaration:

```
function SetThreadManager(const NewTM: TThreadManager;
                        var OldTM: TThreadManager) : Boolean
function SetThreadManager(const NewTM: TThreadManager) : Boolean
```

Visibility: default

Description: `SetThreadManager` sets the thread manager to `NewTM`. If `OldTM` is given, `SetThreadManager` uses it to return the previously used thread manager.

The function returns `True` if the threadmanager was set successfully, `False` if an error occurred.

For more information about thread programming, see the programmer's guide.

Errors: If an error occurred cleaning up the previous manager, or an error occurred initializing the new manager, `False` is returned.

See also: `GetThreadManager` ([1477](#)), `TThreadManager` ([1400](#))

75.12.313 SetUnicodeStringManager

Synopsis: Set the unicodestring manager.

Declaration:

```
procedure SetUnicodeStringManager(const &New: TUnicodeStringManager)
procedure SetUnicodeStringManager(const &New: TUnicodeStringManager;
                                var Old: TUnicodeStringManager)
```

Visibility: default

Description: `SetUnicodeStringManager` sets the current unicodestring manager to `New`. Optionally, it returns the currently active widestring manager in `Old`.

UnicodeStrings are implemented in different ways on different platforms. Therefore, the Free Pascal Runtime library has no fixed implementation of widestring routines. Instead, it defines a Unicode-String manager record, with callbacks that can be set to an implementation which is most efficient on the current platform. On windows, standard Windows routines will be used. On Unix and Linux, an implementation based on the C library is available (in unit `cwstring`).

It is possible to implement a custom unicodestring manager, optimized for the current application, without having to recompile the complete Run-Time Library.

See also: `TUnicodeStringManager` ([1403](#))

75.12.314 SetVariantManager

Synopsis: Set the current variant manager.

Declaration:

```
procedure SetVariantManager(const VarMgr: tvariantmanager)
```

Visibility: default

Description: `SetVariantManager` sets the variant manager to `varmgr`.

See also: `GetVariantManager` ([1478](#))

75.12.315 SetWideStringManager

Synopsis: Set the widestring manager.

Declaration: `procedure SetWideStringManager(const &New: TUnicodeStringManager)`
`procedure SetWideStringManager(const &New: TUnicodeStringManager;`
`var Old: TUnicodeStringManager)`

Visibility: default

Description: `SetWideStringManager` sets the current widestring manager to `New`. Optionally, it returns the currently active widestring manager in `Old`.

WideStrings are implemented in different ways on different platforms. Therefore, the Free Pascal Runtime library has no fixed implementation of widestring routines. Instead, it defines a WideString manager record, with callbacks that can be set to an implementation which is most efficient on the current platform. On windows, standard Windows routines will be used. On Unix and Linux, an implementation based on the C library is available (in unit `cwstring`).

It is possible to implement a custom widestring manager, optimized for the current application, without having to recompile the complete Run-Time Library.

See also: `TWideStringManager` ([1412](#))

75.12.316 ShortCompareText

Synopsis: Compare 2 shortstrings.

Declaration: `function ShortCompareText(const S1: shortstring; const S2: shortstring)`
`: SizeInt`

Visibility: default

Description: `ShortCompareText` compares two shortstrings, `S1` and `S2`, and returns the following result:

<0 if `S1`<`S2`.

0 if `S1`=`S2`.

>0 if `S1`>`S2`.

The comparison of the two strings is case-insensitive. The function does not take internationalization settings into account, it simply compares ASCII values.

Errors: None.

See also: `CompareText` ([1669](#))

75.12.317 Sin

Synopsis: Calculate sine of angle.

Declaration: `function Sin(d: ValReal) : ValReal`

Visibility: default

Description: `Sin` returns the sine of its argument `X`, where `X` is an angle in radians. If the absolute value of the argument is larger than $2\hat{6}3$, then the result is undefined.

Errors: None.

See also: Cos ([1446](#)), Pi ([1514](#)), Exp ([1462](#)), Ln ([1502](#))

Listing: ./examples/refex/ex62.pp

Program Example62;

{ Program to demonstrate the Sin function. }

begin

Writeln (Sin(Pi):0:1); *{ Prints 0.0 }*

Writeln (Sin(Pi/2):0:1); *{ Prints 1.0 }*

end.

75.12.318 SizeOf

Synopsis: Return size (in bytes) of a variable or type.

Declaration: function SizeOf(X: TAnyType) : LongInt

Visibility: default

Description: SizeOf returns the size, in bytes, of any variable or type-identifier.

Remark This isn't really a RTL function. Its result is calculated at compile-time, and hard-coded in the executable.

Errors: None.

See also: Addr ([1420](#)), BitSizeOf ([1335](#))

Listing: ./examples/refex/ex63.pp

Program Example63;

{ Program to demonstrate the SizeOf function. }

Var

 I : Longint;

 S : **String** [10];

begin

Writeln (SizeOf(I)); *{ Prints 4 }*

Writeln (SizeOf(S)); *{ Prints 11 }*

end.

75.12.319 SizeofResource

Synopsis: Return the size of a particular resource.

Declaration: function SizeofResource(ModuleHandle: TFPResourceHMODULE;
 ResHandle: TFPResourceHandle) : LongWord

Visibility: default

Description: SizeOfResource returns the size of the resource identified by ResHandle in module identified by ModuleHandle. ResHandle should be obtained from a call to LoadResource ([1503](#))

Errors: In case of an error, 0 is returned.

See also: FindResource ([1468](#)), FreeResource ([1472](#)), LoadResource ([1503](#)), LockResource ([1504](#)), UnlockResource ([1567](#)), FreeResource ([1472](#))

75.12.320 Slice

Synopsis: Return part of an array.

Declaration: `function Slice(const A: ArrayType; ACount: Integer) : ArrayType2`

Visibility: default

Description: `Slice` returns the first `ACount` elements from the array `A`. It returns an array with the same element type as `A`, but this array is not assignment compatible to any other array, and can therefor only be used in open array arguments to functions.

See also: `Length` ([1500](#)), `SetLength` ([1538](#))

Listing: `./examples/refex/ex113.pp`

Program Example113;

{ Program to demonstrate the Slice function. }

```

procedure ShowArray(const A: array of Integer);
var
    I: Integer;
begin
    for I := Low(A) to High(A) do
        WriteLn(I, ' ', A[I]);
    end;

begin
    ShowArray(Slice([1,2,3,4],2));
end.
```

75.12.321 Space

Synopsis: Return a string of spaces.

Declaration: `function Space(b: Byte) : shortstring`

Visibility: default

Description: `Space` returns a shortstring with length `B`, consisting of spaces.

See also: `StringOfChar` ([1552](#))

75.12.322 SPtr

Synopsis: Return current stack pointer.

Declaration: `function SPtr : Pointer`

Visibility: default

Description: `Sptr` returns the current stack pointer.

Errors: None.

See also: `SSeg` ([1549](#))

Listing: `./examples/refex/ex64.pp`

```

program Example64;

{ Program to demonstrate the sptr function. }

var p:pointer;

begin
  p:=sptr; { P Contains now the current stack position. }
end.

```

75.12.323 Sqr

Synopsis: Calculate the square of a value.

Declaration: `function Sqr(l: LongInt) : LongInt`
`function Sqr(l: Int64) : Int64`
`function Sqr(l: QWord) : QWord`
`function Sqr(d: ValReal) : ValReal`

Visibility: default

Description: `Sqr` returns the square of its argument X.

Errors: None.

See also: `Sqrt` ([1548](#)), `Ln` ([1502](#)), `Exp` ([1462](#))

Listing: ./examples/refex/ex65.pp

```

Program Example65;

{ Program to demonstrate the Sqr function. }
Var i : Integer;

begin
  For i:=1 to 10 do
    writeln (Sqr(i):3);
end.

```

75.12.324 Sqrt

Synopsis: Calculate the square root of a value.

Declaration: `function Sqrt(d: ValReal) : ValReal`

Visibility: default

Description: `Sqrt` returns the square root of its argument X, which must be positive.

Errors: If X is negative, then a run-time error is generated.

See also: `Sqr` ([1548](#)), `Ln` ([1502](#)), `Exp` ([1462](#))

Listing: ./examples/refex/ex66.pp

Program Example66;

{ Program to demonstrate the Sqrt function. }

```
begin
  Writeln (Sqrt(4):0:3); { Prints 2.000 }
  Writeln (Sqrt(2):0:3); { Prints 1.414 }
end.
```

75.12.325 SSeg

Synopsis: Return stack segment register value.

Declaration: `function SSeg : Word`

Visibility: default

Description: `SSeg` returns the Stack Segment. This function is only supported for compatibility reasons, as `Sptr` returns the correct contents of the stackpointer.

Errors: None.

See also: `Sptr` ([1547](#))

Listing: `./examples/refex/ex67.pp`

Program Example67;

{ Program to demonstrate the SSeg function. }

Var W : Longint;

```
begin
  W:=SSeg;
end.
```

75.12.326 StackTop

Synopsis: Top location of the stack.

Declaration: `function StackTop : Pointer`

Visibility: default

Description: `StackTop` contains the top of the stack for the current process. It is used to check the heap on some operating systems, and is set by the system unit initialization code. Do not use or modify this value.

See also: `StackBottom` ([1417](#)), `StackLength` ([1417](#))

75.12.327 Str

Synopsis: Convert a numerical or enumeration value to a string.

Declaration: `procedure Str(var X: TNumericType; var S: string)`

Visibility: default

Description: `Str` returns a string which represents the value of `X`. `X` can be any numerical or enumerated type. The actual declaration of `Str` is not according to pascal syntax, and should be

```
procedure Str(const X: TNumericType[:NumPlaces[:Decimals]];var S: String)
```

Where the optional `NumPlaces` and `Decimals` specifiers control the formatting of the string: `NumPlaces` gives the total width of the string, and `Decimals` the number of decimals after the decimal separator char.

`Str` can also be used to convert an enumerated type value to a string representation of the declared enumeration value. That means that the following will work:

```
Type
  TMyEnum = (One,Two);

Var
  S : String;
begin
  Str(One,S);
  Writeln(S);
end.
```

This will write `One` on the screen, which is consistent with the following - equivalent - program:

```
Type
  TMyEnum = (One,Two);

Var
  S : String;
  E : TMyEnum;
begin
  E:=One;
  Str(E,S);
  Writeln(S);
end.
```

For scoped enumerated types, only the value is written, which means the following program will have the same output:

```
{ $SCOPEDENUMS+ }
Type
  TMyEnum = (One,Two);

Var
  S : String;

begin
  Str(One,S);
  Writeln(S);
end.
```

Errors: None.

See also: [Val \(1570\)](#)

Listing: ./examples/refex/ex68.pp

Program Example68;

```
{ Program to demonstrate the Str function. }
```

```
Var S : String;
```

```
Function IntToStr (I : Longint) : String;
```

```
Var S : String;
```

```
begin
```

```
  Str (I,S);
```

```
  IntToStr:=S;
```

```
end;
```

```
begin
```

```
  S:=''+IntToStr(-233)+'';
```

```
  Writeln (S);
```

```
end.
```

75.12.328 StringCodePage

Synopsis: Get the code page of a string.

```
Declaration: function StringCodePage(const S: RawByteString) : TSystemCodePage
              ; Overload
              function StringCodePage(const S: UnicodeString) : TSystemCodePage
              ; Overload
```

Visibility: default

Description: `StringCodePage` returns the code page of a string (`S`), regardless of the string type. It accesses the internal structures of the string to retrieve this information. For an empty string, `DefaultSystemCodePage` ([1415](#)) is returned.

See also: `DefaultSystemCodePage` ([1415](#)), `StringElementSize` ([1551](#)), `StringRefCount` ([1552](#)), `SetCodePage` ([1537](#))

75.12.329 StringElementSize

Synopsis: Get the character size of a string.

```
Declaration: function StringElementSize(const S: RawByteString) : Word; Overload
              function StringElementSize(const S: UnicodeString) : Word; Overload
```

Visibility: default

Description: `StringCodePage` returns the character size of a string (`S`), regardless of the string type. It accesses the internal structures of the string to retrieve this information. For an empty string, `SizeOf (AnsiChar)` (normally 1) is returned.

See also: `StringCodePage` ([1551](#)), `StringRefCount` ([1552](#))

75.12.330 StringOfChar

Synopsis: Return a string consisting of 1 character repeated N times.

```
Declaration: function StringOfChar(c: AnsiChar; l: SizeInt) : AnsiString
             function StringOfChar(c: UnicodeChar; l: SizeInt) : UnicodeString
```

Visibility: default

Description: `StringOfChar` creates a new `String` of length `l` and fills it with the character `c`.

It is equivalent to the following calls:

```
SetLength(StringOfChar, 1);
FillChar(Pointer(StringOfChar)^, Length(StringOfChar), c);
```

Errors: None.

See also: [SetLength \(1538\)](#)

Listing: ./examples/refex/ex97.pp

Program Example97;

 $\{ \$H+ \}$

{ Program to demonstrate the StringOfChar function. }

```
Var S : String;
```

```
begin
    S:=StringOfChar(' ',40)+'Aligned at column 41.';
    Writeln(s);
end.
```

75.12.331 StringRefCount

Synopsis: Get the reference count of a string.

```
Declaration: function StringRefCount(const S: RawByteString) : SizeInt; Overload
             function StringRefCount(const S: UnicodeString) : SizeInt; Overload
```

Visibility: default

Description: `StringRefCount` returns the reference count of a string (S), regardless of the string type. It accesses the internal structures of the string to retrieve this information. For an empty string, 0 is returned.

See also: [StringCodePage \(1551\)](#), [StringElementSize \(1551\)](#)

75.12.332 StringToPPChar

Synopsis: Split string in list of null-terminated strings.

```
Declaration: function StringToPPChar(var S: AnsiString; ReserveEntries: Integer)
                : PPChar
                function StringToPPChar(S: PChar; ReserveEntries: Integer) : PPChar
```

Visibility: default

Description: `StringToPPChar` splits the string `S` in words, replacing any whitespace with zero characters. It returns a pointer to an array of `pchars` that point to the first letters of the words in `S`. This array is terminated by a `Nil` pointer.

The function does *not* add a zero character to the end of the string unless it ends on whitespace.

The function reserves memory on the heap to store the array of `PChar`; The caller is responsible for freeing this memory.

This function is only available on certain platforms.

Errors: None.

See also: `ArrayStringToPPchar` ([1422](#))

75.12.333 StringToUnicodeChar

Synopsis: Convert an ansistring to a null-terminated array of Unicode characters.

Declaration: `function StringToUnicodeChar(const Src: RawByteString;
Dest: PUnicodeChar; DestSize: SizeInt)
: PUnicodeChar`

Visibility: default

Description: `StringToUnicodeChar` converts the ansistring `S` to a unicodestring and places the result in `Dest`. The size of the memory location pointed to by `Dest` must be given in `DestSize`. If the result string is longer than the available size, the result string will be truncated.

The function always returns `Dest`.

Errors: No check is performed to see whether `Dest` points to a valid memory location.

See also: `UnicodeCharToString` ([1565](#)), `UnicodeCharLenToString` ([1564](#))

75.12.334 StringToWideChar

Synopsis: Convert a string to an array of widechars.

Declaration: `function StringToWideChar(const Src: RawByteString; Dest: PWideChar;
DestSize: SizeInt) : PWideChar`

Visibility: default

Description: `StringToWideChar` converts a single-byte-character string `Src` to a null-terminated array of `WideChars`. The destination for this array is pointed to by `Dest`, and contains room for at least `DestSize` widechars.

Errors: No validity checking is performed on `Dest`.

See also: `WideCharToString` ([1573](#)), `WideCharToStrVar` ([1574](#)), `WideCharLenToStrVar` ([1573](#)), `WideCharLenToString` ([1573](#))

75.12.335 StrLen

Synopsis: Length of a null-terminated string.

Declaration: `function StrLen(p: PChar) : SizeInt`

Visibility: default

Description: Returns the length of the null-terminated string P.

Errors: None.

75.12.336 StrPas

Synopsis: Convert a null-terminated string to a shortstring.

Declaration: `function StrPas(p: PChar) : shortstring`

Visibility: default

Description: Converts a null terminated string in P to a Pascal string, and returns this string. The string is truncated at 255 characters.

Errors: None.

75.12.337 subtract(variant,variant):variant

Synopsis: Implement subtraction (−) operation on variants.

Declaration: `operator -(const op1: variant; const op2: variant) : variant`

Visibility: default

Description: The implementation of the subtraction − operation is delegated to the variant manager with operation `opSubtract`.

Errors: Execution of this operator may result in an exception if no variant manager is installed or if the types of the operand are not suitable for the operation.

See also: operator `-(variant, variant): variant` ([1335](#))

75.12.338 Succ

Synopsis: Return next element of ordinal type.

Declaration: `function Succ(X: TOrdinal) : TOrdinal`

Visibility: default

Description: `Succ` returns the element that succeeds the element that was passed to it. If it is applied to the last value of the ordinal type, and the program was compiled with range checking on (`{ $R+ }`), then a run-time error will be generated.

for an example, see `Ord` ([1512](#)).

You cannot use this function on enumerations with assigned values.

Errors: Run-time error 201 is generated when the result is out of range.

See also: `Ord` ([1512](#)), `Pred` ([1517](#)), `High` ([1483](#)), `Low` ([1506](#))

75.12.339 SuspendThread

Synopsis: Suspend a running thread.

Declaration: `function SuspendThread(threadHandle: TThreadID) : DWord`

Visibility: default

Description: `SuspendThread` suspends a running thread. The thread is identified with its handle or ID `threadHandle`.

The function returns zero if successful. A nonzero return value indicates failure.

Errors: If a failure occurred, a nonzero result is returned. The meaning is system dependent.

See also: `ResumeThread` ([1525](#)), `KillThread` ([1499](#))

75.12.340 Swap

Synopsis: Swap high and low bytes/words of a variable.

Declaration: `function Swap(X: Word) : Word`
`function Swap(X: Integer) : Integer`
`function Swap(X: LongInt) : LongInt`
`function Swap(X: Cardinal) : Cardinal`
`function Swap(X: QWord) : QWord`
`function Swap(X: Int64) : Int64`

Visibility: default

Description: `Swap` swaps the high and low order bytes of `X` if `X` is of type `Word` or `Integer`, or swaps the high and low order words of `X` if `X` is of type `Longint` or `Cardinal`. The return type is the type of `X`

Errors: None.

See also: `Lo` ([1502](#)), `Hi` ([1482](#))

Listing: `./examples/refex/ex69.pp`

Program Example69;

```
{ Program to demonstrate the Swap function. }
Var W : Word;
    L : Longint;

begin
  W:=$1234;
  W:=Swap(W);
  if W<>$3412 then
    writeln ('Error when swapping word !');
  L:=$12345678;
  L:=Swap(L);
  if L<>$56781234 then
    writeln ('Error when swapping Longint !');
end.
```

75.12.341 SwapEndian

Synopsis: Swap endianness of the argument.

Declaration: `function SwapEndian(const AValue: SmallInt) : SmallInt`
`function SwapEndian(const AValue: Word) : Word`
`function SwapEndian(const AValue: LongInt) : LongInt`
`function SwapEndian(const AValue: DWord) : DWord`
`function SwapEndian(const AValue: Int64) : Int64`
`function SwapEndian(const AValue: QWord) : QWord`

Visibility: default

Description: `SwapEndian` will swap the endianness of the bytes in its argument.

Errors: None.

See also: `hi` ([1482](#)), `lo` ([1502](#)), `swap` ([1555](#)), `BEToN` ([1430](#)), `NToBE` ([1510](#)), `NToLE` ([1510](#)), `LEToN` ([1501](#))

75.12.342 SysAllocMem

Synopsis: System memory manager: Allocate memory.

Declaration: `function SysAllocMem(size: PtrUInt) : Pointer`

Visibility: default

Description: `SysFreeMemSize` is the system memory manager implementation for `AllocMem` ([1421](#))

See also: `AllocMem` ([1421](#))

75.12.343 SysAssert

Synopsis: Standard Assert failure implementation.

Declaration: `procedure SysAssert(const Msg: ShortString; const FName: ShortString;`
`LineNo: LongInt; ErrorAddr: Pointer)`

Visibility: default

Description: `SysAssert` is the standard implementation of the assertion failed code. It is the default value of the `AssertErrorProc` constant. It will print the assert message `Msg` together with the filename `FName` and linenumber `LineNo` to standard error output (`StdErr`) and will halt the program with exit code 227. The error address `ErrorAddr` is ignored.

See also: `AssertErrorProc` ([1342](#))

75.12.344 SysBacktraceStr

Synopsis: Format an address suitable for inclusion in a backtrace.

Declaration: `function SysBacktraceStr(Addr: CodePointer) : ShortString`

Visibility: default

Description: `SysBackTraceStr` will create a string representation of the address `Addr`, suitable for inclusion in a stack backtrace.

Errors: None.

75.12.345 SysFlushStdIO

Synopsis: Flush all standard IO file descriptors.

Declaration: `procedure SysFlushStdIO`

Visibility: default

Description: `SysFlushStdIO` calls `flush` (1469) on all standard file descriptors: `output` (1417), `stdout` (1417), `stderr` (1417), `erroutput` (1416)

See also: `output` (1417), `stdout` (1417), `stderr` (1417), `erroutput` (1416), `flush` (1469)

75.12.346 SysFreemem

Synopsis: System memory manager free routine.

Declaration: `function SysFreemem(p: pointer) : PtrUInt`

Visibility: default

Description: `SysFreeem` is the system memory manager implementation for `FreeMem` (1471)

See also: `FreeMem` (1471)

75.12.347 SysFreememSize

Synopsis: System memory manager free routine.

Declaration: `function SysFreememSize(p: pointer; Size: PtrUInt) : PtrUInt`

Visibility: default

Description: `SysFreeemSize` is the system memory manager implementation for `FreeMem` (1471)

See also: `MemSize` (1507)

75.12.348 SysGetFPCHeapStatus

Synopsis: Return the status of the FPC heapmanager.

Declaration: `function SysGetFPCHeapStatus : TFPCHeapStatus`

Visibility: default

Description: `SysGetFPCHeapStatus` returns the status of the default FPC heapmanager. It is set as the default value of the corresponding `GetFPCHeapStatus` (1474) function.

Errors: None. The result of this function is bogus information if the current heapmanager is not the standard FPC heapmanager.

See also: `GetFPCHeapStatus` (1474)

75.12.349 SysGetHeapStatus

Synopsis: System implementation of GetHeapStatus ([1474](#)).

Declaration: `function SysGetHeapStatus : THeapStatus`

Visibility: default

Description: `SysGetHeapStatus` is the system implementation of the `GetHeapStatus` ([1474](#)) call.

See also: `GetHeapStatus` ([1474](#))

75.12.350 SysGetmem

Synopsis: System memory manager memory allocator.

Declaration: `function SysGetmem(Size: PtrUInt) : Pointer`

Visibility: default

Description: `SysGetmem` is the system memory manager implementation for `GetMem` ([1474](#))

See also: `GetMem` ([1474](#)), `GetMemory` ([1475](#))

75.12.351 SysInitExceptions

Synopsis: Initialize exceptions.

Declaration: `procedure SysInitExceptions`

Visibility: default

Description: `SysInitExceptions` initializes the exception system. This procedure should never be called directly, it is taken care of by the RTL.

75.12.352 SysInitFPU

Synopsis: Initialize the FPU.

Declaration: `procedure SysInitFPU`

Visibility: default

Description: `SysInitFPU` initializes (resets) the floating point unit, if one is available. It is called for instance when a new thread is started.

See also: `BeginThread` ([1430](#))

75.12.353 SysInitStdIO

Synopsis: Initialize standard input and output.

Declaration: `procedure SysInitStdIO`

Visibility: default

Description: `SysInitStdIO` initializes the standard input and output files: `Output` ([1417](#)), `Input` ([1416](#)) and `StdErr` ([1417](#)). This routine is called by the initialization code of the system unit, there should be no need to call it directly.

75.12.354 SysMemSize

Synopsis: System memory manager: free size.

Declaration: `function SysMemSize(p: pointer) : PtrUInt`

Visibility: default

Description: `SysFreeMemSize` is the system memory manager implementation for `MemSize` ([1507](#))

See also: `MemSize` ([1507](#))

75.12.355 SysReAllocMem

Synopsis: System memory manager: Reallocate memory.

Declaration: `function SysReAllocMem(var p: pointer; size: PtrUInt) : Pointer`

Visibility: default

Description: `SysReallocMem` is a help routine for the system memory manager implementation for `ReAllocMem` ([1523](#)).

See also: `ReAllocMem` ([1523](#))

75.12.356 SysResetFPU

Synopsis: Reset the floating point unit.

Declaration: `procedure SysResetFPU`

Visibility: default

Description: `SysResetFPU` resets the floating point unit. There should normally be no need to call this unit; the compiler itself takes care of this.

75.12.357 SysSetCtrlBreakHandler

Synopsis: System CTRL-C handler.

Declaration: `function SysSetCtrlBreakHandler(Handler: TCtrlBreakHandler)
: TCtrlBreakHandler`

Visibility: default

Description: `SysSetCtrlBreakHandler` sets the CTRL-C handler to the `Handler` callback, and returns the previous value of the handler.

See also: `TCtrlBreakHandler` ([1386](#))

75.12.358 SysTryResizeMem

Synopsis: System memory manager: attempt to resize memory.

Declaration: `function SysTryResizeMem(var p: pointer; size: PtrUInt) : Boolean`

Visibility: default

Description: `SysTryResizeMem` is a help routine for the system memory manager implementation for `ReAllocMem` ([1523](#)), `SysReAllocMem` ([1559](#))

See also: `SysReAllocMem` ([1559](#)), `ReAllocMem` ([1523](#))

75.12.359 ThreadGetPriority

Synopsis: Return the priority of a thread.

Declaration: `function ThreadGetPriority(threadHandle: TThreadID) : LongInt`

Visibility: default

Description: `ThreadGetPriority` returns the priority of thread `TThreadID` to `Prio`. The returned priority is a value between -15 and 15.

Errors: None.

See also: `ThreadSetPriority` ([1560](#))

75.12.360 ThreadSetPriority

Synopsis: Set the priority of a thread.

Declaration: `function ThreadSetPriority(threadHandle: TThreadID; Prio: LongInt) : Boolean`

Visibility: default

Description: `ThreadSetPriority` sets the priority of thread `TThreadID` to `Prio`. Priority is a value between -15 and 15.

Errors: None.

See also: `ThreadGetPriority` ([1560](#))

75.12.361 ThreadSwitch

Synopsis: Signal possibility of thread switch.

Declaration: `procedure ThreadSwitch`

Visibility: default

Description: `ThreadSwitch` signals the operating system that the thread should be suspended and that another thread should be executed.

This call is a hint only, and may be ignored.

See also: `SuspendThread` ([1555](#)), `ResumeThread` ([1525](#)), `KillThread` ([1499](#))

75.12.362 ToSingleByteFileSystemEncodedFileName

Synopsis: Convert string to encoding for use in single-byte file system API.

Declaration: `function ToSingleByteFileSystemEncodedFileName(const Str: UnicodeString) : RawByteString`
`function ToSingleByteFileSystemEncodedFileName (const arr: Array of WideChar) : RawByteString`
`function ToSingleByteFileSystemEncodedFileName(const Str: RawByteString) : RawByteString`

Visibility: default

Description: `ToSingleByteFileSystemEncodedFileName` converts the argument (`Str` or `Arr`) to a single-byte string, encoded using the codepage used by the single-byte file system API.

This routine is simply an auxiliary routine, which converts the argument to a single-byte string using `DefaultFileSystemCodePage` (1414) as a codepage.

See also: `DefaultFileSystemCodePage` (1414)

75.12.363 Trunc

Synopsis: Truncate a floating point value.

Declaration: `function Trunc(d: ValReal) : Int64`

Visibility: default

Description: `Trunc` returns the integer part of `X`, which is always smaller than (or equal to) `X` in absolute value.

Errors: An invalid floating point operation error may occur in one of several conditions:

- `Infinity` or `NegInfinity` is specified.
- `Nan` is specified.
- `X` is out of the valid range of integers.

See also: `Frac` (1471), `Int` (1493), `Round` (1529)

Listing: `./examples/refex/ex70.pp`

Program Example70;

{ Program to demonstrate the Trunc function. }

```
begin
  Writeln (Trunc(123.456)); { Prints 123 }
  Writeln (Trunc(-123.456)); { Prints -123 }
  Writeln (Trunc(12.3456)); { Prints 12 }
  Writeln (Trunc(-12.3456)); { Prints -12 }
end.
```

Listing: `./examples/refex/ex124.pp`

program ex124;

```
{
  Example to show various error conditions for the Trunc() function.
  SysUtils is used to convert the runtime error to an exception that can be caught
  Math is used to have access to Nan and (Neg)Infinity
}
```

```
{ $mode objfpc }
{ $h+ }
uses
```

```
  sysutils, math;
var
  D: Double;
  Q: QWord;
```

```

    L: Int64;
begin
    D := 2.0 * High(QWord);
    try
        Q := Trunc(D);
    except
        on E: Exception do writeln(E.ClassName, ': ', E.Message); //EInvalidOp: Invalid floating point operation
    end;
    D := NaN;
    try
        Q := Trunc(D);
    except
        on E: Exception do writeln(E.ClassName, ': ', E.Message); //EInvalidOp: Invalid floating point operation
    end;
    D := Infinity;
    try
        Q := Trunc(D);
    except
        on E: Exception do writeln(E.ClassName, ': ', E.Message); //EInvalidOp: Invalid floating point operation
    end;
    D := NegInfinity;
    try
        Q := Trunc(D);
    except
        on E: Exception do writeln(E.ClassName, ': ', E.Message); //EInvalidOp: Invalid floating point operation
    end;
    D := -2.0 * High(QWord);
    try
        L := Trunc(D);
    except
        on E: Exception do writeln(E.ClassName, ': ', E.Message); //EInvalidOp: Invalid floating point operation
    end;
end.

```

75.12.364 Truncate

Synopsis: Truncate the file at position.

Declaration: `procedure Truncate(var F: File)`

Visibility: default

Description: `Truncate` truncates the (opened) file `F` at the current file position.

Errors: Depending on the state of the `{SI}` switch, a runtime error can be generated if there is an error. In the `{SI-}` state, use `IOResult` to check for errors.

See also: [Append \(1421\)](#), [Filepos \(1463\)](#), [Seek \(1534\)](#)

Listing: `./examples/refex/ex71.pp`

Program Example71;

{ Program to demonstrate the Truncate function. }

```

Var F : File of longint;
    I, L : Longint;

```

```

begin
  Assign (F,'test.tmp');
  Rewrite (F);
  For I:=1 to 10 Do
    Write (F,I);
  Writeln ('Filesize before Truncate : ',FileSize(F));
  Close (f);
  Reset (F);
  Repeat
    Read (F,I);
  Until i=5;
  Truncate (F);
  Writeln ('Filesize after Truncate : ',FileSize(F));
  Close (f);
end.

```

75.12.365 TryEnterCriticalSection

Synopsis: Try entering a critical section.

Declaration: `function TryEnterCriticalSection(var cs: TRTLCRITICALSECTION) : LongInt`

Visibility: default

Description: `TryEnterCriticalSection` attempts to enter critical section `cs`. It returns at once. The return value is zero if another thread owns the critical section, or nonzero if the current thread already owns or successfully obtained the critical section.

75.12.366 TypeInfo

Synopsis: Return pointer to type information for type.

Declaration: `function TypeInfo(const T: AnyType) : Pointer`

Visibility: default

Description: `TypeInfo` is a compiler intrinsic: it returns a pointer to the generated type information (RTTI) for the type `T`. If no type information was yet generated for the type, this statement will ensure that type information is available, i.e. the result is always non-nil.

See also: [Default \(1448\)](#), [TypeOf \(1563\)](#), [GetTypeKind \(1477\)](#), [Initialize \(1489\)](#), [Finalize \(1467\)](#)

75.12.367 TypeOf

Synopsis: Return pointer to VMT of an object.

Declaration: `function TypeOf(T: TObjectType) : Pointer`

Visibility: default

Description: `TypeOf` is a compiler intrinsic: it returns a pointer to the VMT of the object type `T`.

See also: [Default \(1448\)](#), [TypeInfo \(1563\)](#), [GetTypeKind \(1477\)](#)

75.12.368 UCS4StringToUnicodeString

Synopsis: Convert a UCS-4 encoded string to a Unicode string.

Declaration: `function UCS4StringToUnicodeString(const s: UCS4String) : UnicodeString`

Visibility: default

Description: `UCS4StringToUnicodeString` converts the UCS-4 encoded string `S` to a Unicode string and returns the resulting string.

This function requires the widestring manager.

See also: `UnicodeStringToUCS4String` ([1565](#))

75.12.369 UCS4StringToWideString

Synopsis:

Declaration: `function UCS4StringToWideString(const s: UCS4String) : WideString`

Visibility: default

Description:

75.12.370 Unassigned

Synopsis: Unassigned variant.

Declaration: `function Unassigned : Variant`

Visibility: default

75.12.371 UnicodeCharLenToString

Synopsis: Convert a memory buffer with Unicode characters to a unicodestring.

Declaration: `function UnicodeCharLenToString(S: PUnicodeChar; Len: SizeInt)
: UnicodeString`

Visibility: default

Description: `UnicodeCharLenToString` converts the Unicode characters in buffer `S` with at most `len` bytes length, to a unicodestring and returns the result.

This function requires the use of a widestring manager.

Errors: No checking is done to see if the pointer `S` or length `len` are valid.

See also: `StringToUnicodeChar` ([1553](#)), `UnicodeCharToString` ([1565](#))

75.12.372 UnicodeCharLenToStrVar

Synopsis: Convert a memory buffer with Unicode characters to an ansistring.

Declaration: `procedure UnicodeCharLenToStrVar(Src: PUnicodeChar; Len: SizeInt;
out Dest: UnicodeString)
procedure UnicodeCharLenToStrVar(Src: PUnicodeChar; Len: SizeInt;
out Dest: AnsiString)`

Visibility: default

Description: `UnicodeCharLenToString` converts the Unicode characters in buffer `S` with at most `len` bytes length, to an ansistring and returns the result in `Dest`

This function does the same as `UnicodeCharLenToString` (1564).

Errors: No checking is done to see if the pointer `S` or length `len` are valid.

See also: `StringToUnicodeChar` (1553), `UnicodeCharToString` (1565), `UnicodeCharLenToString` (1564), `UnicodeCharToStrVar` (1565)

75.12.373 UnicodeCharToString

Synopsis: Convert Unicode character to string.

Declaration: `function UnicodeCharToString(S: PUnicodeChar) : UnicodeString`

Visibility: default

Description: `UnicodeCharToString` converts a null-word-terminated array of Unicode characters in `S` to a Unicode string value. It simply calls `UnicodeCharLenToString` (1564) with the length of the string `S`.

This function requires the use of a widestring manager.

Errors: No checking is done to see if the pointer `S` is valid.

See also: `StringToUnicodeChar` (1553), `UnicodeCharLenToString` (1564), `WidestringManager` (1418)

75.12.374 UnicodeCharToStrVar

Synopsis: Convert a null-terminated memory buffer with Unicode characters to an ansistring.

Declaration: `procedure UnicodeCharToStrVar(S: PUnicodeChar; out Dest: AnsiString)`

Visibility: default

Description: `UnicodeCharLenToString` converts the Unicode characters in buffer `S` up to the first null word, to an ansistring and returns the result in `Dest`

This function does the same as `UnicodeCharToString` (1565).

Errors: No checking is done to see if the pointer `S` is valid.

See also: `StringToUnicodeChar` (1553), `UnicodeCharToString` (1565), `UnicodeCharLenToString` (1564), `UnicodeCharToString` (1565)

75.12.375 UnicodeStringToUCS4String

Synopsis: Convert a Unicode string to a UCS-4 string.

Declaration: `function UnicodeStringToUCS4String(const s: UnicodeString) : UCS4String`

Visibility: default

Description: `UnicodeStringToUCS4String` converts a Unicode string `S` to a UCS-4 encoded string, and returns the resulting string.

This function requires the widestring manager.

See also: `UCS4StringToUnicodeString` (1564)

75.12.376 UnicodeToUtf8

Synopsis: Convert a unicode (UTF16) string to a UTF8 string

Declaration: `function UnicodeToUtf8 (Dest: PChar; Source: PUnicodeChar;
 MaxBytes: SizeInt) : SizeInt
 function UnicodeToUtf8 (Dest: PChar; MaxDestBytes: SizeUInt;
 Source: PUnicodeChar; SourceChars: SizeUInt)
 : SizeUInt`

Visibility: default

Description: `Utf8ToUnicode` converts the buffer in `Source` with a length of `SourceBytes` or for a maximum length of `MaxBytes` (or `MaxDestbytes`) bytes to the buffer pointed to by `Dest`. If `Dest` is `Nil` it will return the number of bytes needed to convert the UTF16 to UTF8. if `MaxDestBytes` allows for a null terminator, then it is also added to the output.

The function returns the number of copied bytes, or, if `Dest` is `Nil`, then it will return the number of bytes needed to perform the conversion.

Errors: On error, -1 is returned.

See also: `Utf8ToUnicode` ([1570](#)), `UTF8Encode` ([1569](#)), `UTF8Decode` ([1568](#)), `Utf8ToAnsi` ([1569](#)), `SetWideStringManager` ([1545](#))

75.12.377 UniqueString

Synopsis: Make sure reference count of string is 1.

Declaration: `procedure UniqueString (var S: RawByteString)
 procedure UniqueString (var S: UnicodeString)
 procedure UniqueString (var S: WideString)`

Visibility: default

Description: `UniqueString` ensures that the ansistring `S` has reference count 1. It makes a copy of `S` if this is necessary, and returns the copy in `S`

Errors: None.

75.12.378 UnloadLibrary

Synopsis: Unload a previously loaded library.

Declaration: `function UnloadLibrary (Lib: TLibHandle) : Boolean`

Visibility: default

Description: `UnloadLibrary` unloads a previously loaded library (specified by the handle `lib`). The call returns `True` if successful, `False` otherwise.

Errors: On error, `False` is returned.

See also: `LoadLibrary` ([1503](#)), `GetProcAddress` ([1476](#))

75.12.379 UnlockResource

Synopsis: Unlock a previously locked resource.

Declaration: `function UnlockResource(ResData: TFPResourceHGLOBAL) : LongBool`

Visibility: default

Description: `UnlockResource` unlocks a previously locked resource. Note that this function does not exist on windows, it's only needed on other platforms.

Errors: The function returns `False` if it failed.

See also: `FindResource` ([1468](#)), `FreeResource` ([1472](#)), `SizeofResource` ([1546](#)), `LoadResource` ([1503](#)), `lockResource` ([1504](#)), `FreeResource` ([1472](#))

75.12.380 UnPack

Synopsis: Create unpacked array from packed array.

Declaration: `procedure UnPack(const Z: PackedArrayType; out A: UnpackedArrayType; StartIndex: TIndexType)`

Visibility: default

Description: `UnPack` will copy the elements of a packed array (Z) to an unpacked array (A). All elements in Z are copied to A, starting at index `StartIndex` in A. The type of the index variable `StartIndex` must match the type of the index of A.

Obviously, the type of the elements of the arrays A and Z must match.

See also: `Pack` ([1513](#))

75.12.381 UpCase

Synopsis: Convert a string to all uppercase.

Declaration: `function UpCase(const s: shortstring) : shortstring`
`function UpCase(c: Char) : Char`
`function UpCase(const s: ansistring) : ansistring`
`function UpCase(const s: UnicodeString) : UnicodeString`
`function UpCase(c: UnicodeChar) : UnicodeChar`
`function UpCase(const s: WideString) : WideString`

Visibility: default

Description: `UpCase` returns the uppercase version of its argument C. If its argument is a string, then the complete string is converted to uppercase. The type of the returned value is the same as the type of the argument.

`UpCase` does not change the number of characters in the string.

Errors: None.

See also: `Lowercase` ([1506](#))

Listing: `./examples/refex/ex72.pp`

```

program Example72;

{ Program to demonstrate the upcase function. }

var c:char;

begin
  for c:='a' to 'z' do
    write(upcase(c));
  Writeln;
  { This doesn't work in TP, but it does in Free Pascal }
  Writeln(upcase('abcdefghijklmnopqrstuvwxyz'));
end.

```

75.12.382 Utf8CodePointLen

Synopsis: Length of an UTF-8 codepoint.

Declaration: `function Utf8CodePointLen(P: PAnsiChar; MaxLookAhead: SizeInt; IncludeCombiningDiacriticalMarks: Boolean) : SizeInt`

Visibility: default

Description: `Utf8CodePointLen` returns the length of the UTF-8 codepoint starting at the beginning of `P`. It will look at at most `MaxLookAhead` bytes to do create this codepoint. If `IncludeCombiningDiacriticalMarks` is true, combining diacritical marks trailing the first codepoint (which itself can also be such a mark) will be considered to be part of the codepoint.

If the function returns a value > 0 , then this is the number of bytes occupied by the codepoint and, if requested, the trailing combining diacritical marks. If the result $= 0$, this means that all bytes within the requested `MaxLookAhead` could be part of a single valid codepoint and, if requested, its trailing diacritical marks, but that the codepoint is incomplete and more bytes need to be looked at. If the result is < 0 , then the function determined that the codepoint was invalid after processing the number of bytes equal to the absolute value of the function result.

If `IncludeCombiningDiacriticalMarks` is True, then

- If the function processes all `MaxLookAhead` bytes, it will return the value `MaxLookAhead` rather than 0, even though in theory more combining diacritical marks might follow if more bytes would be looked at. Therefore, in order to ascertain that all combining diacritical marks are processed, pass all bytes at once to this function.
- If an invalid sequence is detected while processing a potential combining diacritical mark after a valid codepoint has been found already, the function will return the length of this valid codepoint (plus that of any preceding valid combining diacritical marks) as a positive value. The idea is that this invalid sequence at the end is by definition not a combining diacritical mark (since all of those are valid sequences) and hence should not render the preceding codepoint invalid.

Errors: None.

75.12.383 UTF8Decode

Synopsis: Convert an UTF-8 encoded ansistring to a unicodestring.

Declaration: `function UTF8Decode(const s: RawByteString) : UnicodeString`

Visibility: default

Description: `UTF8Decode` converts the UTF-8 encoded ansistring `S` to a unicodestring and returns the resulting string. It calls the low-level `Utf8ToUnicode` (1570) function to do the actual work.

See also: `UTF8Encode` (1569), `Utf8ToAnsi` (1569), `SetWideStringManager` (1545), `Utf8ToUnicode` (1570)

75.12.384 UTF8Encode

Synopsis: Convert a widestring or unicodestring to an UTF-8 encoded ansistring.

Declaration: `function UTF8Encode(const s: RawByteString) : RawByteString`
`function UTF8Encode(const s: UnicodeString) : RawByteString`
`function UTF8Encode(const s: WideString) : RawByteString`

Visibility: default

Description: `UTF8Encode` converts an ansistring or widestring `S` to the equivalent UTF-8 encoded Unicode string and returns this resulting string. It calls the low-level `UnicodeToUTF8` (1566) function to do the actual work.

The resulting string has code page `CP_UTF8`.

See also: `UTF8Decode` (1568), `Utf8ToAnsi` (1569), `UnicodeToUtf8` (1566), `SetWideStringManager` (1545)

75.12.385 Utf8ToAnsi

Synopsis: Convert a UTF-8 encoded Unicode string to an ansistring.

Declaration: `function Utf8ToAnsi(const s: RawByteString) : RawByteString`

Visibility: default

Description: `Utf8ToAnsi` converts an utf8-encode Unicode string to an ansistring. It converts the string to a widestring and then converts the widestring to an ansistring.

For this function to work, a widestring manager must be installed.

See also: `UTF8Encode` (1569), `UTF8Decode` (1568), `SetWideStringManager` (1545)

75.12.386 UTF8ToString

Synopsis: Convert UTF8 to Unicode String.

Declaration: `function UTF8ToString(const s: RawByteString) : UnicodeString`
`function UTF8ToString(const S: ShortString) : unicodestring`
`function UTF8ToString(const S: PAnsiChar) : unicodestring`
`function UTF8ToString(const S: Array of AnsiChar) : unicodestring`
`function UTF8ToString(const S: Array of Byte) : unicodestring`

Visibility: default

Description: `UTF8ToString` converts a UTF8 single byte string to a UTF16 encoded unicodestring. The source string `S` is interpreted as an array of UTF8 characters.

75.12.387 Utf8ToUnicode

Synopsis: Convert a buffer with UTF-8 characters to widestring characters.

Declaration: `function Utf8ToUnicode(Dest: PUnicodeChar; Source: PChar;
MaxChars: SizeInt) : SizeInt
function Utf8ToUnicode(Dest: PUnicodeChar; MaxDestChars: SizeUInt;
Source: PChar; SourceBytes: SizeUInt) : SizeUInt`

Visibility: default

Description: `Utf8ToUnicode` converts the buffer in `Source` with a length of `SourceBytes` or for a maximum length of `MaxChars` (or `MaxDestChars`) widestring characters to the buffer pointed to by `Dest`. If `Dest` is `Nil` it will return the number of widechars needed to convert the UTF8. if `MaxDestChars` allows for a null terminator, then it is also added to the output.

The function returns the number of copied widestring characters, or, if `Dest` is `Nil`, then it will return the number of characters needed to perform the conversion.

Errors: On error, -1 is returned.

See also: `UnicodeToUTF8` ([1566](#)), `UTF8Encode` ([1569](#)), `UTF8Decode` ([1568](#)), `Utf8ToAnsi` ([1569](#)), `SetWideStringManager` ([1545](#))

75.12.388 Val

Synopsis: Calculate numerical/enumerated value of a string.

Declaration: `procedure Val(const S: string; var V; var Code: Word)`

Visibility: default

Description: `Val` converts the value represented in the string `S` to a numerical value or an enumerated value, and stores this value in the variable `V`, which can be of any integer type (signed or unsigned, sizes from 1 to 8 bytes), floating point type (of any supported size) or any enumerated type. If the conversion is not successful, then the parameter `Code` contains the index of the character in `S` which prevented the conversion. The string `S` is allowed to contain spaces in the beginning.

The string `S` can contain a number in decimal, hexadecimal, binary or octal format, as described in the language reference. For enumerated values, the string must be the name of the enumerated value. The name is searched case insensitively.

The string can use various radices in addition to a decimal (radix 10) notation:

- \$, 0x or x can be used for hexadecimal notation. Allowed characters are 0 . . 9 and A . . F (case insensitive)
- & for octal notation. Allowed characters are 0 . . 7.
- % for binary notation. Allowed characters are 0 . . 1.

The conversion to enumerated exists only as of version 2.3.1 (or later) of the compiler.

If `S` represents a numerical value in decimal notation, or a negative number, then the numerical value must fit into the range of the type of `V`, otherwise `Val` will report an error.

If `S` represents a numerical value in binary, octal or hexadecimal notation, and does not contain a minus sign, then the numerical value must fit into the unsigned counterpart of the type of `V` and will be casted to the type of `V`. If not, `Val` will report an error.

Errors: If the conversion doesn't succeed, the value of `Code` indicates the position where the conversion went wrong. The value of `V` is then undefined.

See also: Str ([1549](#))

Listing: ./examples/refex/ex74.pp

Program Example74;

```
{ Program to demonstrate the Val function. }
Var I, Code : Integer;

begin
  Val (ParamStr (1),I,Code);
  If Code<>0 then
    Writeln ('Error at position ',code,' : ',Paramstr(1)[Code])
  else
    Writeln ('Value : ',I);
end.
```

75.12.389 VarArrayGet

Synopsis: Get a value from a single cell of a variant array.

Declaration: `function VarArrayGet (const A: Variant; const Indices: Array of LongInt) : Variant`

Visibility: default

Description: VarArrayGet returns the value in the variant array A at the location indicated by Indices. Thus the statement

```
B:=VarArrayGet (A, [2, 1]);
```

is equivalent to

```
B:=A [2, 1];
```

The difference is that the previous is usable when the amount of indices is not known at compile time.

Errors: If the number of indices is wrong (or out of range) an exception may be raised.

See also: VarArrayPut ([1571](#))

75.12.390 VarArrayPut

Synopsis: Put a value in a single cell of a variant array.

Declaration: `procedure VarArrayPut (var A: Variant; const Value: Variant; const Indices: Array of LongInt)`

Visibility: default

Description: VarArrayPut puts Value in the variant array A at the location indicated by Indices. Thus the statement

```
VarArrayPut (A, B, [2, 1]);
```

is equivalent to

```
A[2,1] := B;
```

The difference is that the previous is usable when the amount of indices is not known at compile time.

Errors: If the number of indices is wrong (or out of range) an exception may be raised.

See also: `VarArrayGet` ([1571](#))

75.12.391 `VarArrayRedim`

Synopsis: Redimension a variant array.

Declaration: `procedure VarArrayRedim(var A: Variant; HighBound: SizeInt)`

Visibility: default

Description: `VarArrayRedim` re-sizes the first dimension of the variant array `A`, giving it a new high bound `HighBound`. Obviously, `A` must be a variant array for this function to work.

75.12.392 `VarCast`

Synopsis: Cast a variant to a certain type.

Declaration: `procedure VarCast(var dest: variant; const source: variant;
vartype: LongInt)`

Visibility: default

Description: `VarCast` converts the variant in `Source` to the type indicated in `VarType` and returns the result in `dest`. The `VarType` must be one of the predefined `VarNNN` constants.

Errors: If the conversion is not possible because the value cannot be correctly casted, then a run-time error or an exception may occur.

75.12.393 `WaitForThreadTerminate`

Synopsis: Wait for a thread to terminate.

Declaration: `function WaitForThreadTerminate(threadHandle: TThreadID;
TimeoutMs: LongInt) : DWord`

Visibility: default

Description: `WaitForThreadTerminate` waits for a thread to finish its execution. The thread is identified by its handle or ID `threadHandle`. If the thread does not exit within `TimeoutMs` milliseconds, the function will return with an error value.

The function returns the exit code of the thread.

Not all platforms support the timeout parameter: the Unix platforms (with threads support based on pthreads) do not support timeout, and will wait indefinitely for the thread to exit.

See also: `EndThread` ([1455](#)), `KillThread` ([1499](#))

75.12.394 WideCharLenToString

Synopsis: Convert a length-limited array of widechar to an unicodestring.

Declaration: `function WideCharLenToString(S: PWideChar; Len: SizeInt) : UnicodeString`

Visibility: default

Description: `WideCharLenToString` converts at most `Len` widecharacters from the null-terminated widechar array `S` to an unicodestring, and returns the unicodestring.

Errors: No validity checking is performed on `S`. Passing an invalid pointer may lead to access violations.

See also: `StringToWideChar` ([1553](#)), `WideCharToString` ([1573](#)), `WideCharToStrVar` ([1574](#)), `WideCharLenToStrVar` ([1573](#))

75.12.395 WideCharLenToStrVar

Synopsis: Convert a length-limited array of widechar to an ansistring.

Declaration: `procedure WideCharLenToStrVar(Src: PWideChar; Len: SizeInt;
out Dest: UnicodeString)
procedure WideCharLenToStrVar(Src: PWideChar; Len: SizeInt;
out Dest: AnsiString)`

Visibility: default

Description: `WideCharLenToString` converts at most `Len` widecharacters from the null-terminated widechar array `Src` to an ansistring or Unicode string, and returns the resulting in `Dest`.

Errors: No validity checking is performed on `Src`. Passing an invalid pointer may lead to access violations.

See also: `StringToWideChar` ([1553](#)), `WideCharToString` ([1573](#)), `WideCharToStrVar` ([1574](#)), `WideCharLenToString` ([1573](#))

75.12.396 WideCharToString

Synopsis: Convert a null-terminated array of widechar to an unicodestring.

Declaration: `function WideCharToString(S: PWideChar) : UnicodeString`

Visibility: default

Description: `WideCharToString` converts the null-terminated widechar array `S` to an unicodestring, and returns the unicodestring.

Errors: No validity checking is performed on `Src`. Passing an invalid pointer, or an improperly terminated array may lead to access violations.

See also: `StringToWideChar` ([1553](#)), `WideCharToStrVar` ([1574](#)), `WideCharLenToStrVar` ([1573](#)), `WideCharLenToString` ([1573](#))

75.12.397 WideCharToStrVar

Synopsis: Convert a null-terminated array of widechar to an ansistring.

Declaration: `procedure WideCharToStrVar(S: PWideChar; out Dest: UnicodeString)`
`procedure WideCharToStrVar(S: PWideChar; out Dest: AnsiString)`

Visibility: default

Description: `WideCharToString` converts the null-terminated widechar array `S` to an ansistring or Unicode string, and returns the resulting string in `Dest`.

Errors: No validity checking is performed on `S`. Passing an invalid pointer, or an improperly terminated array may lead to access violations.

See also: `StringToWideChar` (1553), `WideCharToString` (1573), `WideCharLenToStrVar` (1573), `WideCharLenToString` (1573)

75.12.398 WideStringToUCS4String

Synopsis: Convert a widestring to a UCS-4 encoded string.

Declaration: `function WideStringToUCS4String(const s: WideString) : UCS4String`

Visibility: default

Description: Convert a widestring to a UCS-4 encoded string.

75.12.399 Write

Synopsis: Write variable to a text file or standard output.

Declaration: `procedure Write(V1: Type1)`
`procedure Write(V1: Type1; V2: type2)`
`procedure Write(V1: Type1; V2: Type2; V3: Type3)`
`procedure Write(var F: Text; V1: Type1)`
`procedure Write(var F: Text; V1: Type1; V2: type2)`
`procedure Write(var F: Text; V1: Type1; V2: Type2; V3: Type3)`

Visibility: default

Description: `Write` writes the contents of the variables `V1`, `V2`, `V3` etc. to the file `F`. `F` can be a typed file, or a Text file. If `F` is a typed file, then the variables `V1`, `V2` etc. must be of the same type as the type in the declaration of `F`. Untyped files are not allowed.

The `Write` command accepts an arbitrary number of arguments. The `V1`, `V2`, `V3` in the declaration here are in fact just samples, the actual number may be much higher. The types of arguments (`Type1` etc.) are limited to the following types:

- Any character type.
- Any string type (including `pchar`).
- Any ordinal type (integer, enumerated).
- The `Int64` and `QWord` type.
- Any floating-point type (such as `double`, `single`, `extended`).

If the parameter `F` is omitted, standard output is assumed. If `F` is of type `Text`, then the necessary conversions are done such that the output of the variables is in human-readable format. This conversion is done for all numerical types. Strings are printed exactly as they are in memory, as well as `PChar` types.

The format of the numerical conversions can be influenced through the following modifiers: `OutputVariable: NumChars [: Decimals ]` This will print the value of `OutputVariable` with a minimum of `NumChars` characters, from which `Decimals` are reserved for the decimals. If the number cannot be represented with `NumChars` characters, `NumChars` will be increased, until the representation fits. If the representation requires less than `NumChars` characters then the output is filled up with spaces, to the left of the generated string, thus resulting in a right-aligned representation. If no formatting is specified, then the number is written using its natural length, with nothing in front of it if it's positive, and a minus sign if it's negative. Real numbers are, by default, written in scientific notation.

Remark When writing string variables, no codepage conversions are done. The string is copied as-is to the file descriptor. In particular, for console output, it is the programmer's responsibility to make sure that the codepage of the string matches the codepage of the console.

Remark Note that on MS Windows GUI applications do not have a standard output by default: Standard file descriptors are available only when the

```
{ $APPTYPE CONSOLE }
```

directive is present in the program file.

Errors: If an error occurs, a run-time error is generated. This behavior can be controlled with the `{ $I }` switch.

See also: [WriteLn \(1575\)](#), [Read \(1519\)](#), [ReadLn \(1521\)](#), [Blockwrite \(1432\)](#)

75.12.400 WriteBarrier

Synopsis: Memory write barrier.

Declaration: `procedure WriteBarrier`

Visibility: `default`

Description: `WriteBarrier` is a low-level instruction to force a write barrier in the CPU: write (store) operations before and after the barrier are separate.

See also: [ReadBarrier \(1520\)](#), [ReadDependencyBarrier \(1520\)](#), [ReadWriteBarrier \(1522\)](#)

75.12.401 WriteLn

Synopsis: Write variable to a text file or standard output and append newline.

Declaration: `procedure WriteLn(V1: Type1)`
`procedure WriteLn(V1: Type1; V2: type2)`
`procedure WriteLn(V1: Type1; V2: Type2; V3: Type3)`
`procedure WriteLn(var F: Text; V1: Type1)`
`procedure WriteLn(var F: Text; V1: Type1; V2: type2)`
`procedure WriteLn(var F: Text; V1: Type1; V2: Type2; V3: Type3)`

Visibility: `default`

Description: `WriteLn` does the same as `Write` (1574) for text files, and emits a Carriage Return - LineFeed character pair after that. If the parameter `F` is omitted, standard output is assumed. If no variables are specified, a newline character sequence is emitted, resulting in a new line in the file `F`.

Remark The newline character is determined by the `slinebreak` (1361) constant.

Remark When writing string variables, no codepage conversions are done. The string is copied as-is to the file descriptor. In particular, for console output, it is the programmer's responsibility to make sure that the codepage of the string matches the codepage of the console.

More details can be found in the `Write` (1574) description.

Errors: If an error occurs, a run-time error is generated. This behavior can be controlled with the `{ $I }` switch.

See also: `Write` (1574), `Read` (1519), `ReadLn` (1521), `Blockwrite` (1432), `slinebreak` (1361)

Listing: `./examples/refex/ex75.pp`

Program `Example75;`

{ Program to demonstrate the Write(Ln) function. }

Var

`F : File of Longint;`

`L : Longint;`

begin

`Write ('This is on the first line ! '); { No CR/LF pair! }`

`Writeln ('And this too...');`

`Writeln ('But this is already on the second line...');`

`Assign (f,'test.tmp');`

`Rewrite (f);`

`For L:=1 to 10 do`

`write (F,L); { No writeln allowed here ! }`

`Close (f);`

end.

75.12.402 WriteStr

Synopsis: Write variables to a string.

Declaration: `procedure WriteStr(out S: string; Args: Arguments)`

Visibility: default

Description: `WriteStr` behaves like `Write` (1574), except that it stores its output in the string variable `S` instead of a file. Semantically, the `WriteStr` call is equivalent to writing the arguments to a file using the `Write` call, and then reading them into `S` using the `Read` call from the same file:

`var`

`F : Text;`

begin

`Rewrite (F) ;`

`Write (F,Args) ;`

`Close (F) ;`

`Reset (F) ;`

`Read (F,S) ;`

```

    Close(F);
end;

```

Obviously, the `WriteStr` call does not use a temporary file.

`WriteStr` is defined in the ISO Extended Pascal standard. More information on the allowed arguments and the possible formatting can be found in the description of `Write` ([1574](#)).

See also: `Write` ([1574](#)), `ReadStr` ([1521](#)), `Read` ([1519](#))

75.13 TDoubleRec

```

TDoubleRec = packed record
private
    Bias = $3FF;
    function GetExp
    : QWord;
    procedure SetExp(e: QWord);
    function GetSign : Boolean
    ;
    procedure SetSign(s: Boolean);
    function GetFrac : QWord;
    procedure
    SetFrac(e: QWord);
public
    function Mantissa(IncludeHiddenBit: Boolean
    ) : QWord;
    function Fraction : ValReal;
    function Exponent : LongInt
    ;
    property Sign : Boolean;
    property Exp : QWord;
    property Frac
    : QWord;
    function SpecialType : TFloatSpecial;
    procedure BuildUp
    (const _Sign: Boolean; const _Mantissa: QWord;
    const _Exponent: LongInt);
case Byte of
0: (
public
    Bytes : Array
    [0..7] of Byte;
);
1: (
public
    Words : Array[0..3] of Word;
);
2
: (
public
    Data : QWord;
);

```

```

3: (
public
  Value : Double;
);
end

```

TDoubleRec models the memory layout of a double value when using software floating point math.

75.13.1 Method overview

Page	Method	Description
1579	BuildUp	Build a double value.
1579	Exponent	Exponent of the floating point value.
1578	Fraction	Fraction of the floating point value.
1578	Mantissa	Mantissa of the floating point value.
1579	SpecialType	Is the floating point value special ?

75.13.2 Property overview

Page	Properties	Access	Description
1579	Exp	rw	Exponent bitpattern representation.
1580	Frac	rw	Fractional part of double.
1579	Sign	rw	Sign of the floating point value.

75.13.3 TDoubleRec.Mantissa

Synopsis: Mantissa of the floating point value.

Declaration: `function Mantissa(IncludeHiddenBit: Boolean) : QWord`

Visibility: public

Description: `Mantissa` returns the Mantissa part (significand bitpattern without hidden bit) of the floating point value.

See also: `TDoubleRec.Fraction` ([1578](#)), `TDoubleRec.Exponent` ([1579](#)), `TDoubleRec.SpecialType` ([1579](#))

75.13.4 TDoubleRec.Fraction

Synopsis: Fraction of the floating point value.

Declaration: `function Fraction : ValReal`

Visibility: public

Description: `Fraction` returns the fraction (value after decimal) of the floating point value.

See also: `TDoubleRec.Mantissa` ([1578](#)), `TDoubleRec.Exponent` ([1579](#)), `TDoubleRec.SpecialType` ([1579](#))

75.13.5 TDoubleRec.Exponent

Synopsis: Exponent of the floating point value.

Declaration: `function Exponent : LongInt`

Visibility: public

Description: `Exponent` returns the exponent (X in $m \times 2^X$ representation) of the floating point value.

See also: `TDoubleRec.Fraction` ([1578](#)), `TDoubleRec.Mantissa` ([1578](#)), `TDoubleRec.SpecialType` ([1579](#))

75.13.6 TDoubleRec.SpecialType

Synopsis: Is the floating point value special ?

Declaration: `function SpecialType : TFloatSpecial`

Visibility: public

Description: `SpecialType` returns special characteristics of the floating point value, if any. See `TFloatSpecial` ([1390](#)) for an explanation of the various special values.

See also: `TDoubleRec.Fraction` ([1578](#)), `TDoubleRec.Mantissa` ([1578](#)), `TDoubleRec.Exponent` ([1579](#)), `TFloatSpecial` ([1390](#))

75.13.7 TDoubleRec.BuildUp

Synopsis: Build a double value.

Declaration: `procedure BuildUp(const _Sign: Boolean; const _Mantissa: QWord;
const _Exponent: LongInt)`

Visibility: public

Description: `Buildup` will build a double value from the given `_Sign`, `_Mantissa` and `_Exponent`.

See also: `TDoubleRec.Mantissa` ([1578](#)), `TDoubleRec.Sign` ([1579](#)), `TDoubleRec.Exp` ([1579](#))

75.13.8 TDoubleRec.Sign

Synopsis: Sign of the floating point value.

Declaration: `Property Sign : Boolean`

Visibility: public

Access: Read,Write

75.13.9 TDoubleRec.Exp

Synopsis: Exponent bitpattern representation.

Declaration: `Property Exp : QWord`

Visibility: public

Access: Read,Write

Description: `Exp` returns the internal bit representation of the exponent of the floating point value.

See also: `TDoubleRec.Sign` ([1579](#)), `TDoubleRec.Exponent` ([1579](#))

75.13.10 TDoubleRec.Frac

Synopsis: Fractional part of double.

Declaration: `Property Frac : QWord`

Visibility: `public`

Access: `Read,Write`

Description: `Frac` is the fractional part of the `Double`. This is the bit pattern representing the fractional part including the preceding 1. (The mantissa is that bit pattern without the preceding 1)

See also: `Frac` ([1471](#))

75.14 TExtended80Rec

```
TExtended80Rec = packed record
private
  Bias = $3FFF;
  function
  GetExp : QWord;
  procedure SetExp(e: QWord);
  function GetSign
    : Boolean;
  procedure SetSign(s: Boolean);
public
  function Mantissa
    (IncludeHiddenBit: Boolean) : QWord;
  function Fraction : Extended
  ;
  function Exponent : LongInt;
  property Sign : Boolean;
  property
  Exp : QWord;
  function SpecialType : TFloatSpecial;
  procedure
  BuildUp(const _Sign: Boolean; const _Mantissa: QWord;
    const _Exponent: LongInt);
case Byte of
0: (
public
  Bytes
    : Array[0..9] of Byte;
);
1: (
public
  Words : Array[0..4] of Word
  ;
);
2: (
public
  _Exp : Word;
  Frac : QWord;
);
```

```

3: (
public
  Value
  : Extended;
);
end

```

TExtended80Rec models the memory layout of an extended value when using software floating point math.

75.14.1 Method overview

Page	Method	Description
1582	BuildUp	
1582	Exponent	Exponent of the floating point value.
1581	Fraction	Fraction of the floating point value.
1581	Mantissa	Mantissa of the floating point value.
1582	SpecialType	Is the floating point value special ?

75.14.2 Property overview

Page	Properties	Access	Description
1582	Exp	rw	Exponent representation.
1582	Sign	rw	Sign of the floating point value.

75.14.3 TExtended80Rec.Mantissa

Synopsis: Mantissa of the floating point value.

Declaration: `function Mantissa(IncludeHiddenBit: Boolean) : QWord`

Visibility: public

Description: Mantissa returns the Mantissa part (bit pattern of the significand, skipping the hidden bit) of the floating point value.

See also: TExtended80Rec.Fraction ([1581](#)), TExtended80Rec.Exponent ([1582](#)), TExtended80Rec.SpecialType ([1582](#))

75.14.4 TExtended80Rec.Fraction

Synopsis: Fraction of the floating point value.

Declaration: `function Fraction : Extended`

Visibility: public

Description: Fraction returns the fraction (value after decimal) of the floating point value.

See also: TExtended80Rec.Mantissa ([1581](#)), TExtended80Rec.Exponent ([1582](#)), TExtended80Rec.SpecialType ([1582](#))

75.14.5 TExtended80Rec.Exponent

Synopsis: Exponent of the floating point value.

Declaration: `function Exponent : LongInt`

Visibility: public

Description: `Exponent` returns the exponent (the X in the $m \times 2^X$ representation) of the floating point value.

See also: `TExtended80Rec.Fraction` (1581), `TExtended80Rec.Mantissa` (1581), `TExtended80Rec.SpecialType` (1582)

75.14.6 TExtended80Rec.SpecialType

Synopsis: Is the floating point value special ?

Declaration: `function SpecialType : TFloatSpecial`

Visibility: public

Description: `SpecialType` returns special characteristics of the floating point value, if any. See `TFloatSpecial` (1390) for an explanation of the various special values.

See also: `TExtended80Rec.Fraction` (1581), `TExtended80Rec.Mantissa` (1581), `TExtended80Rec.Exponent` (1582), `TFloatSpecial` (1390)

75.14.7 TExtended80Rec.BuildUp

Declaration: `procedure BuildUp(const _Sign: Boolean; const _Mantissa: QWord;
const _Exponent: LongInt)`

Visibility: public

75.14.8 TExtended80Rec.Sign

Synopsis: Sign of the floating point value.

Declaration: `Property Sign : Boolean`

Visibility: public

Access: Read,Write

75.14.9 TExtended80Rec.Exp

Synopsis: Exponent representation.

Declaration: `Property Exp : QWord`

Visibility: public

Access: Read,Write

Description: `Exponent` returns the internal bit representation of the exponent of the floating point value.

See also: `TExtended80Rec.Sign` (1582), `TExtended80Rec.Exponent` (1582)

75.15 TGuid

```

TGuid = packed record
public
  TGuid.class operator =(const aLeft
    : TGUID; const aRight: TGUID)
    : Boolean;
  TGuid.class operator <>(const aLeft: TGUID; const aRight: TGUID
  )
    : Boolean;
  class function Empty : TGUID
  ; Static;
  Create;
  function IsEmpty : Boolean;
case Integer of
  1: (
public
  Data1 : DWord;
  Data2 : Word;
  Data3 : Word;
  Data4
    : Array[0..7] of Byte;
  );
  2: (
public
  D1 : DWord;
  D2 : Word;
  D3 : Word;
  D4 : Array[0..7] of Byte;
  );
  3: (
public
  time_low
    : DWord;
  time_mid : Word;
  time_hi_and_version : Word;
  clock_seq_hi_and_reserved
    : Byte;
  clock_seq_low : Byte;
  node : Array[0..5] of Byte;
  );
end

```

Standard GUID representation type.

75.15.1 Method overview

Page	Method	Description
1584	Create	
1584	Empty	
1584	equal(TGUID,TGUID):Boolean	
1584	IsEmpty	
1584	notequal(TGUID,TGUID):Boolean	

75.15.2 TGuid.equal(TGUID,TGUID):Boolean

Declaration: TGuid.class operator =(const aLeft: TGUID; const aRight: TGUID)
: Boolean

Visibility: public

75.15.3 TGuid.notequal(TGUID,TGUID):Boolean

Declaration: TGuid.class operator <>(const aLeft: TGUID; const aRight: TGUID)
: Boolean

Visibility: public

75.15.4 TGuid.Empty

Declaration: class function Empty : TGUID; Static

Visibility: public

75.15.5 TGuid.Create

Declaration: class function Create(const aData; aBigEndian: Boolean) : TGUID
; Overload; Static
class function Create(const aData: Array of Byte;
aStartIndex: Cardinal; aBigEndian: Boolean) : TGUID
; Overload; Static
class function Create(const aData: PByte; aBigEndian: Boolean) : TGUID
; Overload; Static

Visibility: public

75.15.6 TGuid.IsEmpty

Declaration: function IsEmpty : Boolean

Visibility: public

75.16 tinterfaceentry

```
tinterfaceentry = record
private
    function GetIID : PGuid;
    function
        GetIIDStr : PShortString;
public
    property IID : PGuid;
    property
        IIDStr : PShortString;
    IIDRef : ^PGuid;
    VTable : Pointer;
case
```

```

Integer of
1: (
public
  IOffset : SizeUInt;
);
2: (
public
  IOffsetAsCodePtr
    : CodePointer;
  IIDStrRef : ^PShortString;
  IType : tinterfaceentrytype
  ;
);
end

```

`tinterfaceentry` is used to store the list of Interfaces of a class. This list is stored as an array of `tinterfaceentry` records.

75.16.1 Property overview

Page	Properties	Access	Description
1585	IID	r	Unique GUID for this interface.
1585	IIDStr	r	Pointer to GUID string. Always assigned for COM.

75.16.2 tinterfaceentry.IID

Synopsis: Unique GUID for this interface.

Declaration: `Property IID : PGuid`

Visibility: `public`

Access: `Read`

75.16.3 tinterfaceentry.IIDStr

Synopsis: Pointer to GUID string. Always assigned for COM.

Declaration: `Property IIDStr : PShortString`

Visibility: `public`

Access: `Read`

75.17 TSingleRec

```

TSingleRec = packed record
private
  Bias = $7F;
  function GetExp
    : QWord;
  procedure SetExp(e: QWord);
  function GetSign : Boolean

```

```

;
procedure SetSign(s: Boolean);
function GetFrac : QWord;
procedure
  SetFrac(e: QWord);
public
  function Mantissa(IncludeHiddenBit: Boolean
  ) : QWord;
  function Fraction : ValReal;
  function Exponent : LongInt
;
  property Sign : Boolean;
  property Exp : QWord;
  property Frac
    : QWord;
  function SpecialType : TFloatSpecial;
  procedure BuildUp
    (const _Sign: Boolean; const _Mantissa: QWord;
     const _Exponent: LongInt);
case Byte of
0: (
public
  Bytes : Array
    [0..3] of Byte;
);
1: (
public
  Words : Array[0..1] of Word;
);
2
  : (
public
  Data : DWord;
);
3: (
public
  Value : Single;
);
end

```

TsingleRec models the memory layout of a double value when using software floating point math.

75.17.1 Method overview

Page	Method	Description
1588	BuildUp	
1587	Exponent	Exponent of the floating point value.
1587	Fraction	Fraction of the floating point value.
1587	Mantissa	Mantissa of the floating point value.
1587	SpecialType	Is the floating point value special ?

75.17.2 Property overview

Page	Properties	Access	Description
1588	Exp	rw	Exponent bitpattern representation.
1588	Frac	rw	Fractional part of single.
1588	Sign	rw	Sign of the floating point value.

75.17.3 TSingleRec.Mantissa

Synopsis: Mantissa of the floating point value.

Declaration: `function Mantissa(IncludeHiddenBit: Boolean) : QWord`

Visibility: public

Description: `Mantissa` returns the Mantissa part (significand bitpattern without hidden bit) of the floating point value.

See also: `TSingleRec.Fraction` ([1587](#)), `TSingleRec.Exponent` ([1587](#)), `TSingleRec.SpecialType` ([1587](#))

75.17.4 TSingleRec.Fraction

Synopsis: Fraction of the floating point value.

Declaration: `function Fraction : ValReal`

Visibility: public

Description: `Fraction` returns the fraction (after decimal) of the floating point value.

See also: `TSingleRec.Mantissa` ([1587](#)), `TSingleRec.Exponent` ([1587](#)), `TSingleRec.SpecialType` ([1587](#))

75.17.5 TSingleRec.Exponent

Synopsis: Exponent of the floating point value.

Declaration: `function Exponent : LongInt`

Visibility: public

Description: `Exponent` returns the exponent (X in $m \times 2^X$ representation) of the floating point value.

See also: `TSingleRec.Fraction` ([1587](#)), `TSingleRec.Mantissa` ([1587](#)), `TSingleRec.SpecialType` ([1587](#))

75.17.6 TSingleRec.SpecialType

Synopsis: Is the floating point value special ?

Declaration: `function SpecialType : TFloatSpecial`

Visibility: public

Description: `SpecialType` returns special characteristics of the floating point value, if any. See `TFloatSpecial` ([1390](#)) for an explanation of the various special values.

See also: `TSingleRec.Fraction` ([1587](#)), `TSingleRec.Mantissa` ([1587](#)), `TSingleRec.Exponent` ([1587](#)), `TFloatSpecial` ([1390](#))

75.17.7 TSingleRec.BuildUp

Declaration: `procedure BuildUp(const _Sign: Boolean; const _Mantissa: QWord;
const _Exponent: LongInt)`

Visibility: public

75.17.8 TSingleRec.Sign

Synopsis: Sign of the floating point value.

Declaration: `Property Sign : Boolean`

Visibility: public

Access: Read,Write

75.17.9 TSingleRec.Exp

Synopsis: Exponent bitpattern representation.

Declaration: `Property Exp : QWord`

Visibility: public

Access: Read,Write

Description: `Exp` returns the internal bit representation of the exponent of the floating point value.

See also: `TSingleRec.Sign` ([1588](#)), `TSingleRec.Exponent` ([1587](#))

75.17.10 TSingleRec.Frac

Synopsis: Fractional part of single.

Declaration: `Property Frac : QWord`

Visibility: public

Access: Read,Write

Description: `Frac` is the fractional part of the `Single`. This is the bit pattern representing the fractional part including the preceding 1. (The mantissa is that bit pattern without the preceding 1)

See also: `Frac` ([1471](#))

75.18 TVmt

```
TVmt = record
public
  vInstanceSize : SizeInt;
  vInstanceSize2
  : SizeInt;
  vParentRef : PPVmt;
  vClassName : PShortString;
  vDynamicTable
```

```

: Pointer;
vMethodTable : Pointer;
vFieldTable : Pointer;
vTypeInfo
: Pointer;
vInitTable : Pointer;
vAutoTable : Pointer;
vIntfTable
: pinterfacetable;
vMsgStrPtr : pstringmessagetable;
vDestroy
: CodePointer;
vNewInstance : CodePointer;
vFreeInstance : CodePointer
;
vSafeCallException : CodePointer;
vDefaultHandler : CodePointer
;
vAfterConstruction : CodePointer;
vBeforeDestruction : CodePointer
;
vDefaultHandlerStr : CodePointer;
vDispatch : CodePointer;
vDispatchStr : CodePointer;
vEquals : CodePointer;
vGetHashCode
: CodePointer;
vToString : CodePointer;
private
function GetvParent
: PVmt;
public
property vParent : PVmt;
end

```

TVMT is a record describing the VMT of a class. It's various fields represent the available information in the VMT, as far as it is common to all classes.

75.18.1 Property overview

Page	Properties	Access	Description
1589	vParent	r	Pointer to parent VMT.

75.18.2 TVmt.vParent

Synopsis: Pointer to parent VMT.

Declaration: Property vParent : PVmt

Visibility: public

Access: Read

75.19 IDispatch

75.19.1 Description

IDispatch is the pascal definition of the Windows Dispatch interface definition.

See also: IUnknown ([1593](#))

75.19.2 Method overview

Page	Method	Description
1590	GetIDsOfNames	Return IDs of named procedures.
1590	GetTypeInfo	Return type information about properties.
1590	GetTypeInfoCount	Return number of properties.
1590	Invoke	Invoke a dispatch method.

75.19.3 IDispatch.GetTypeInfoCount

Synopsis: Return number of properties.

Declaration: `function GetTypeInfoCount(out count: LongInt) : HRESULT`

Visibility: default

75.19.4 IDispatch.GetTypeInfo

Synopsis: Return type information about properties.

Declaration: `function GetTypeInfo(Index: LongInt; LocaleID: LongInt; out TypeInfo) : HRESULT`

Visibility: default

75.19.5 IDispatch.GetIDsOfNames

Synopsis: Return IDs of named procedures.

Declaration: `function GetIDsOfNames(const iid: TGuid; names: Pointer; NameCount: LongInt; LocaleID: LongInt; DispIDs: Pointer) : HRESULT`

Visibility: default

Description: Return the ID of a procedure.

75.19.6 IDispatch.Invoke

Synopsis: Invoke a dispatch method.

Declaration: `function Invoke(DispID: LongInt; const iid: TGuid; LocaleID: LongInt; Flags: Word; var params; VarResult: pointer; ExcepInfo: pointer; ArgErr: pointer) : HRESULT`

Visibility: default

75.20 IEnumerable

75.20.1 Description

`IEnumerable` can be used to get an enumerator from a class. If a class implements `IEnumerable`, it can return an enumerator interface `IEnumerator` ([1591](#)).

See also: `IEnumerator` ([1591](#))

75.20.2 Method overview

Page	Method	Description
1591	<code>GetEnumerator</code>	Return an enumerator interface for this class.

75.20.3 IEnumerable.GetEnumerator

Synopsis: Return an enumerator interface for this class.

Declaration: `function GetEnumerator : IEnumerator`

Visibility: default

Description: `GetEnumerator` returns a new `IEnumerator` ([1591](#)) interface for this class. This is called by the compiler whenever a `for in` loop is encountered in the source code to retrieve the enumerator instance.

See also: `IEnumerator` ([1591](#))

75.21 IEnumerator

75.21.1 Description

`IEnumerator` is the interface needed by the `For ... in ...` language construct, when operating on classes. It contains all methods that the compiler needs to implement a loop.

A `for in` loop like the following:

```
For O in MyObject do
  begin
    // do things
  end;
```

is treated by the compiler as equivalent to the following code:

```
Var
  I : IEnumerator;
  O : TObject;

begin
  I:=MyObject.GetEnumerator;
  While I.MoveNext do
    begin
      O:=I.GetCurrent;
      // Do things
```



```

    end;
end.
```

Any class that implements the `IEnumerable` interface must be able to return an `IEnumerator` instance for the compiler to use in a `For in` loop.

See also: `IEnumerable` ([1591](#))

75.21.2 Method overview

Page	Method	Description
1592	<code>GetCurrent</code>	Returns the current element in the iteration cycle.
1592	<code>MoveNext</code>	Move to the next value.
1593	<code>Reset</code>	Reset the pointer.

75.21.3 Property overview

Page	Properties	Access	Description
1593	<code>Current</code>	<code>r</code>	Return the current item.

75.21.4 `IEnumerator.GetCurrent`

Synopsis: Returns the current element in the iteration cycle.

Declaration: `function GetCurrent : TObject`

Visibility: `default`

Description: `getCurrent` should return the object instance representing the current value in the `for in` loop. `GetCurrent` will always be called immediately after `IEnumerator.MoveNext` ([1592](#)) returned `True`.

Remark The actual return type of the interface should not necessarily be `TObject`, it can be any type. The compiler will check the actual return type with the type of the loop variable, and they should match.

See also: `IEnumerator.MoveNext` ([1592](#)), `IEnumerator.Reset` ([1593](#))

75.21.5 `IEnumerator.MoveNext`

Synopsis: Move to the next value.

Declaration: `function MoveNext : Boolean`

Visibility: `default`

Description: `MoveNext` should move the current item pointer to the next available item. It should return `True` if an item is available, `False` if no more items are available. The first time it is called It will be called at the beginning of the `for` loop, so it should position the enumerator on the first value (if there is one). After `MoveNext` has returned `True`, `IEnumerator.GetCurrent` ([1592](#)) will be called to retrieve the item.

See also: `IEnumerator.Reset` ([1593](#)), `IEnumerator.GetCurrent` ([1592](#))

75.21.6 IEnumerator.Reset

Synopsis: Reset the pointer.

Declaration: `procedure Reset`

Visibility: `default`

Description: `Reset` can be implemented to put the pointer at the start of the list. It is not mandatory to implement this method, the compiler does not use it.

See also: `IEnumerator.GetCurrent` ([1592](#)), `IEnumerator.MoveNext` ([1592](#))

75.21.7 IEnumerator.Current

Synopsis: Return the current item.

Declaration: `Property Current : TObject`

Visibility: `default`

Access: `Read`

Description: `Current` simply is the redefinition of `IEnumerator.GetCurrent` ([1592](#)) as a property. It is read-only.

See also: `IEnumerator.GetCurrent` ([1592](#)), `IEnumerator.MoveNext` ([1592](#))

75.22 IInvokable

75.22.1 Description

`IInvokable` is a descendent of `IInterface` ([1372](#)), compiled in the `{ $M+ }` state, so Run-Time Type Information (RTTI) is generated for it.

See also: `IDispatch` ([1590](#)), `IInterface` ([1372](#))

75.23 IUnknown

75.23.1 Description

`IUnknown` is defined by windows. It's the basic interface which all COM objects must implement. The definition does not contain any code.

See also: `IInterface` ([1372](#)), `IDispatch` ([1590](#)), `IInvokable` ([1593](#))

75.23.2 Method overview

Page	Method	Description
1594	<code>QueryInterface</code>	Return pointer to VMT table of interface.
1594	<code>_AddRef</code>	Increase reference count of the interface.
1594	<code>_Release</code>	Decrease reference count of the interface.

75.23.3 IUnknown.QueryInterface

Synopsis: Return pointer to VMT table of interface.

Declaration: `function QueryInterface(const iid: TGuid; out obj) : LongInt`

Visibility: default

75.23.4 IUnknown._AddRef

Synopsis: Increase reference count of the interface.

Declaration: `function _AddRef : LongInt`

Visibility: default

See also: `IUnknown._Release` ([1594](#))

75.23.5 IUnknown._Release

Synopsis: Decrease reference count of the interface.

Declaration: `function _Release : LongInt`

Visibility: default

See also: `IUnknown._AddRef` ([1594](#))

75.24 TAggregatedObject

75.24.1 Description

`TAggregatedObject` implements an object whose lifetime is governed by an external object (or interface). It does not implement the `IUnknown` interface by itself, but delegates all methods to the controller object, as exposed in the `Controller` ([1595](#)) property. In effect, the reference count of the aggregated object is the same as that of its controller, and additionally, all interfaces of the controller are exposed by the aggregated object.

Note that the aggregated object maintains a non-counted reference to the controller.

Aggregated objects should be used when using delegation to implement reference counted objects: the delegated interfaces can be implemented safely by `TAggregatedObject` descendents.

See also: `Create` ([1595](#)), `Controller` ([1595](#))

75.24.2 Method overview

Page	Method	Description
1595	<code>Create</code>	Create a new instance of <code>TAggregatedObject</code> .

75.24.3 Property overview

Page	Properties	Access	Description
1595	<code>Controller</code>	<code>r</code>	Controlling instance.

75.24.4 TAggregatedObject.Create

Synopsis: Create a new instance of TAggregatedObject.

Declaration: `constructor Create(const aController: IUnknown)`

Visibility: public

Description: Create creates a new instance of TAggregatedObject on the heap, and stores a reference to aController, so it can be exposed in the Controller ([1595](#)) property.

Errors: If not enough memory is present on the heap, an exception will be raised. If the aController is Nil, exceptions will occur when any of the TAggregatedObject methods (actually, the IUnknown methods) are used.

See also: Controller ([1595](#))

75.24.5 TAggregatedObject.Controller

Synopsis: Controlling instance.

Declaration: `Property Controller : IUnknown`

Visibility: public

Access: Read

Description: Controller exposes the controlling object, with all interfaces it has.

The value of the controller is set when the TAggregatedObject instance is created.

See also: TAggregatedObject.Create ([1595](#))

75.25 TContainedObject

75.25.1 Description

TContainedObject is the base class for contained objects, i.e. objects that do not implement a reference counting mechanism themselves, but are owned by some other object which handles the reference counting mechanism. It implements the IUnknown interface and, more specifically, the QueryInterface method of IUnknown.

See also: IInterface ([1372](#))

75.25.2 Interfaces overview

Page	Interfaces	Description
1372	IInterface	Basic interface for all COM based interfaces.

75.26 TInterfacedObject

75.26.1 Description

TInterfacedObject is a descendent of TObject ([1597](#)) which implements the IUnknown ([1593](#)) interface. It can be used as a base class for all classes which need reference counting.

See also: IUnknown ([1593](#)), TObject ([1597](#))

75.26.2 Interfaces overview

Page	Interfaces	Description
1593	IUnknown	Basic interface for all COM-based interfaces.

75.26.3 Method overview

Page	Method	Description
1596	AfterConstruction	Handle reference count properly.
1596	BeforeDestruction	Check reference count.
1596	destroy	Destroy interfaced object.
1597	NewInstance	Create a new instance.

75.26.4 Property overview

Page	Properties	Access	Description
1597	RefCount	r	Return the current reference count.

75.26.5 TInterfacedObject.destroy

Synopsis: Destroy interfaced object.

Declaration: `destructor destroy; Override`

Visibility: `public`

Description: `Destroy` overrides the default destructor to clean up: it explicitly sets all counters to zero.

See also: `IUnknown._Release` ([1594](#))

75.26.6 TInterfacedObject.AfterConstruction

Synopsis: Handle reference count properly.

Declaration: `procedure AfterConstruction; Override`

Visibility: `public`

Description: `AfterConstruction` overrides the basic method in `TObject` and adds some additional reference count handling.

Errors: None.

See also: `BeforeDestruction` ([1596](#))

75.26.7 TInterfacedObject.BeforeDestruction

Synopsis: Check reference count.

Declaration: `procedure BeforeDestruction; Override`

Visibility: `public`

Description: `AfterConstruction` overrides the basic method in `TObject` and adds a reference count check: if the reference count is not zero, an error occurs.

Errors: A runtime-error 204 will be generated if the reference count is nonzero when the object is destroyed.

See also: `AfterConstruction` ([1596](#))

75.26.8 TInterfacedObject.NewInstance

Synopsis: Create a new instance.

Declaration: `class function NewInstance : TObject; Override`

Visibility: `public`

Description: `NewInstance` initializes a new instance of `TInterfacedObject` ([1595](#))

Errors: None.

75.26.9 TInterfacedObject.RefCount

Synopsis: Return the current reference count.

Declaration: `Property RefCount : LongInt`

Visibility: `public`

Access: `Read`

Description: `RefCount` returns the current reference count. This reference count cannot be manipulated, except through the methods of `IUnknown` ([1593](#)). When it reaches zero, the class instance is destroyed.

See also: `IUnknown` ([1593](#))

75.27 TObject

75.27.1 Description

`TObject` is the parent root class for all classes in Object Pascal. If a class has no parent class explicitly declared, it is dependent on `TObject`. `TObject` introduces class methods that deal with the class' type information, and contains all necessary methods to create an instance at runtime, and to dispatch messages to the correct method (both string and integer messages).

See also: `TClass` ([1385](#))

75.27.2 Method overview

Page	Method	Description
1604	<code>AfterConstruction</code>	Method called after the constructor was called.
1604	<code>BeforeDestruction</code>	Method called before the destructor is called.
1601	<code>ClassInfo</code>	Return a pointer to the type information for this class.
1601	<code>ClassName</code>	Return the current class name.
1602	<code>ClassNameIs</code>	Check whether the class name equals the given name.
1602	<code>ClassParent</code>	Return the parent class.
1601	<code>ClassType</code>	Return a "class of" pointer for the current class.
1601	<code>CleanupInstance</code>	Finalize the class instance.
1598	<code>Create</code>	<code>TObject</code> Constructor.
1600	<code>DefaultHandler</code>	Default handler for integer message handlers.
1604	<code>DefaultHandlerStr</code>	Default handler for string messages.
1599	<code>Destroy</code>	<code>TObject</code> destructor.
1604	<code>Dispatch</code>	Dispatch an integer message.
1605	<code>DispatchStr</code>	Dispatch a string message.
1607	<code>Equals</code>	Check if two objects are equal.
1603	<code>FieldAddress</code>	Return the address of a field.
1600	<code>Free</code>	Check for <code>Nil</code> and call destructor.
1599	<code>FreeInstance</code>	Clean up instance and free the memory reserved for the instance.
1607	<code>GetHashCode</code>	Return a hash code for the object.
1605	<code>GetInterface</code>	Return a reference to an interface.
1605	<code>GetInterfaceByStr</code>	Return an interface based on its GUID.
1606	<code>GetInterfaceEntry</code>	Return the interface table entry by GUID.
1606	<code>GetInterfaceEntryByStr</code>	Return the interface table entry by string.
1606	<code>GetInterfaceTable</code>	Return a pointer to the table of implemented interfaces for a class.
1606	<code>GetInterfaceWeak</code>	Get a reference to an interface, not increasing the reference count.
1602	<code>InheritsFrom</code>	Check whether class is an ancestor.
1600	<code>InitInstance</code>	Initialize a new class instance.
1602	<code>InstanceSize</code>	Return the size of an instance.
1603	<code>MethodAddress</code>	Return the address of a method.
1603	<code>MethodName</code>	Return the name of a method.
1599	<code>newinstance</code>	Allocate memory on the heap for a new instance.
1607	<code>QualifiedClassName</code>	Fully qualified classname.
1600	<code>SafeCallException</code>	Handle exception object.
1603	<code>StringMessageTable</code>	Return a pointer to the string message table.
1608	<code>ToString</code>	Return a string representation for the object.
1607	<code>UnitName</code>	Unit name.

75.27.3 TObject.Create

Synopsis: `TObject` Constructor.

Declaration: `constructor Create`

Visibility: `public`

Description: `Create` creates a new instance of `TObject`. Currently it does nothing (except allocating memory for the new instance).

Note that allocating the memory for the object instance happens in hidden code generated as part of every constructor, not only in `TObject.Create`. The hidden code calls `NewInstance` ([1599](#)).

See also: [Destroy \(1599\)](#)

75.27.4 TObject.Destroy

Synopsis: TObject destructor.

Declaration: `destructor Destroy; Virtual`

Visibility: public

Description: `Destroy` is the destructor of `TObject`. It will clean up the memory assigned to the instance. Descendent classes should override `destroy` if they want to do additional clean-up. No other destructor should be implemented.

It is bad programming practice to call `Destroy` directly. It is better to call the `Free (1600)` method, because that one will check first if `Self` is different from `Nil`.

To clean up an instance and reset the reference to the instance, it is best to use the `FreeAndNil (1725)` function.

See also: [Create \(1598\)](#), [Free \(1600\)](#)

75.27.5 TObject.newinstance

Synopsis: Allocate memory on the heap for a new instance.

Declaration: `class function newInstance : TObject; Virtual`

Visibility: public

Description: `NewInstance` allocates memory on the heap for a new instance of the current class. If the memory was allocated, the class will be initialized by a call to `InitInstance (1600)`. The function returns the newly initialized instance.

This method can be overridden to implement e.g. object pooling. Note that the method is responsible for correctly initializing the object, i.e. zeroing out the new instance.

Errors: If not enough memory is available, a `Nil` pointer may be returned, or an exception may be raised.

See also: [Create \(1598\)](#), [InitInstance \(1600\)](#), [InstanceSize \(1602\)](#), [FreeInstance \(1599\)](#)

75.27.6 TObject.FreeInstance

Synopsis: Clean up instance and free the memory reserved for the instance.

Declaration: `procedure FreeInstance; Virtual`

Visibility: public

Description: `FreeInstance` cleans up an instance of the current class, and releases the heap memory occupied by the class instance.

See also: [Destroy \(1599\)](#), [InitInstance \(1600\)](#), [NewInstance \(1599\)](#)

75.27.7 TObject.SafeCallException

Synopsis: Handle exception object.

Declaration: `function SafeCallException(exceptobject: TObject;
exceptaddr: CodePointer) : HRESULT; Virtual`

Visibility: public

Description: `SafeCallException` should be overridden to handle exceptions in a method marked with the `savecall` directive. The implementation in `TObject` simply returns zero.

75.27.8 TObject.DefaultHandler

Synopsis: Default handler for integer message handlers.

Declaration: `procedure DefaultHandler(var message); Virtual`

Visibility: public

Description: `DefaultHandler` is the default handler for messages. If a message has an unknown message ID (i.e. does not appear in the table with integer message handlers), then it will be passed to `DefaultHandler` by the `Dispatch` (1604) method.

See also: `Dispatch` (1604), `DefaultHandlerStr` (1604)

75.27.9 TObject.Free

Synopsis: Check for `Nil` and call destructor.

Declaration: `procedure Free`

Visibility: public

Description: `Free` will check the `Self` pointer and calls `Destroy` (1599) if it is different from `Nil`. This is a safer method than calling `Destroy` directly. If a reference to the object must be reset as well (a recommended technique), then the function `FreeAndNil` (1725) should be called.

Errors: None.

See also: `Destroy` (1599), `FreeAndNil` (1725)

75.27.10 TObject.InitInstance

Synopsis: Initialize a new class instance.

Declaration: `class function InitInstance(instance: pointer) : TObject`

Visibility: public

Description: `InitInstance` initializes the memory pointer to by `Instance`. This means that the VMT is initialized, and the interface pointers are set up correctly. The function returns the newly initialized instance.

See also: `NewInstance` (1599), `Create` (1598)

75.27.11 TObject.CleanupInstance

Synopsis: Finalize the class instance.

Declaration: `procedure CleanupInstance`

Visibility: `public`

Description: `CleanUpInstance` finalizes the instance, i.e. takes care of all reference counted objects, by decreasing their reference count by 1, and freeing them if their count reaches zero.

Normally, `CleanupInstance` should never be called, it is called automatically when the object is freed with it's constructor.

Errors: None.

See also: `Destroy` ([1599](#)), `Free` ([1600](#)), `InitInstance` ([1600](#))

75.27.12 TObject.ClassType

Synopsis: Return a "class of" pointer for the current class.

Declaration: `class function ClassType : TClass`

Visibility: `public`

Description: `ClassType` returns a `TClass` ([1385](#)) class type reference for the current class.

See also: `TClass` ([1385](#)), `ClassInfo` ([1601](#)), `ClassName` ([1601](#))

75.27.13 TObject.ClassInfo

Synopsis: Return a pointer to the type information for this class.

Declaration: `class function ClassInfo : pointer`

Visibility: `public`

Description: `ClassInfo` returns a pointer to the type information for this class. This pointer can be used in the various type information routines.

75.27.14 TObject.ClassName

Synopsis: Return the current class name.

Declaration: `class function ClassName : shortstring`

Visibility: `public`

Description: `ClassName` returns the class name for the current class. To check for the class name, you can use the `ClassNameIs` ([1602](#)) class method.

Errors: None.

See also: `ClassInfo` ([1601](#)), `ClassType` ([1601](#)), `ClassNameIs` ([1602](#))

75.27.15 TObject.ClassNameIs

Synopsis: Check whether the class name equals the given name.

Declaration: `class function ClassNameIs(const name: string) : Boolean`

Visibility: public

Description: `ClassNameIs` checks whether `Name` equals the class name. It takes of case sensitivity.

See also: `ClassInfo` ([1601](#)), `ClassType` ([1601](#)), `ClassName` ([1601](#))

75.27.16 TObject.ClassParent

Synopsis: Return the parent class.

Declaration: `class function ClassParent : TClass`

Visibility: public

Description: `ClassParent` returns the class of the parent class of the current class. This is always different from `Nil`, except for `TObject`.

Errors: None.

See also: `ClassInfo` ([1601](#)), `ClassType` ([1601](#)), `ClassNameIs` ([1601](#))

75.27.17 TObject.InstanceSize

Synopsis: Return the size of an instance.

Declaration: `class function InstanceSize : SizeInt`

Visibility: public

Description: `InstanceSize` returns the number of bytes an instance takes in memory. This is Just the memory occupied by the class structure, and does not take into account any additional memory that might be allocated by the constructor of the class.

Errors: None.

See also: `InitInstance` ([1600](#)), `ClassName` ([1601](#)), `ClassInfo` ([1601](#)), `ClassType` ([1601](#))

75.27.18 TObject.InheritsFrom

Synopsis: Check whether class is an ancestor.

Declaration: `class function InheritsFrom(aclass: TClass) : Boolean`

Visibility: public

Description: `InheritsFrom` returns `True` if `AClass` is an ancestor class from the current class, and returns `false` if it is not.

See also: `ClassName` ([1601](#)), `ClassInfo` ([1601](#)), `ClassType` ([1601](#)), `TClass` ([1385](#))

75.27.19 TObject.StringMessageTable

Synopsis: Return a pointer to the string message table.

Declaration: `class function StringMessageTable : pstringmessagetable`

Visibility: public

Description: `StringMessageTable` returns a pointer to the string message table, which can be used to look up methods for dispatching a string message. It is used by the `DispatchStr` (1605) method.

Errors: If there are no string message handlers, nil is returned.

See also: `DispatchStr` (1605), `Dispatch` (1604)

75.27.20 TObject.MethodAddress

Synopsis: Return the address of a method.

Declaration: `class function MethodAddress(const name: shortstring) : CodePointer`

Visibility: public

Description: `MethodAddress` returns the address of a method, searching the method by its name. The `Name` parameter specifies which method should be taken. The search is conducted in a case-insensitive manner.

Errors: If no matching method is found, `Nil` is returned.

See also: `MethodName` (1603), `FieldAddress` (1603)

75.27.21 TObject.MethodName

Synopsis: Return the name of a method.

Declaration: `class function MethodName(address: CodePointer) : shortstring`

Visibility: public

Description: `MethodName` searches the VMT for a method with the specified address and returns the name of the method.

Errors: If no method with the matching address is found, an empty string is returned.

See also: `MethodAddress` (1603), `FieldAddress` (1603)

75.27.22 TObject.FieldAddress

Synopsis: Return the address of a field.

Declaration: `function FieldAddress(const name: shortstring) : pointer`

Visibility: public

Description: `FieldAddress` returns the address of the field with name `name`. The address is the address of the field in the current class instance.

Errors: If no field with the specified name is found, `Nil` is returned.

See also: `MethodAddress` (1603), `MethodName` (1603)

75.27.23 TObject.AfterConstruction

Synopsis: Method called after the constructor was called.

Declaration: `procedure AfterConstruction; Virtual`

Visibility: `public`

Description: `AfterConstruction` is a method called after the constructor was called. It does nothing in the implementation of `TObject` and must be overridden by descendent classes to provide specific behaviour that is executed after the constructor has finished executing. (for instance, call an event handler)

Errors: None.

See also: `BeforeDestruction` ([1604](#)), `Create` ([1598](#))

75.27.24 TObject.BeforeDestruction

Synopsis: Method called before the destructor is called.

Declaration: `procedure BeforeDestruction; Virtual`

Visibility: `public`

Description: `BeforeDestruction` is a method called before the destructor is called. It does nothing in the implementation of `TObject` and must be overridden by descendent classes to provide specific behaviour that is executed before the destructor has finished executing. (for instance, call an event handler)

Errors: None.

See also: `AfterConstruction` ([1604](#)), `Destroy` ([1599](#)), `Free` ([1600](#))

75.27.25 TObject.DefaultHandlerStr

Synopsis: Default handler for string messages.

Declaration: `procedure DefaultHandlerStr(var message); Virtual`

Visibility: `public`

Description: `DefaultHandlerStr` is called for string messages which have no handler associated with them in the string message handler table. The implementation of `DefaultHandlerStr` in `TObject` does nothing and must be overridden by descendent classes to provide specific message handling behaviour.

See also: `DispatchStr` ([1605](#)), `Dispatch` ([1604](#)), `DefaultHandler` ([1600](#))

75.27.26 TObject.Dispatch

Synopsis: Dispatch an integer message.

Declaration: `procedure Dispatch(var message); Virtual`

Visibility: `public`

Description: `Dispatch` looks in the message handler table for a handler that handles `message`. The message is identified by the first dword (cardinal) in the message structure.

If no matching message handler is found, the message is passed to the `DefaultHandler` (1600) method, which can be overridden by descendent classes to add custom handling of messages.

See also: `DispatchStr` (1605), `DefaultHandler` (1600)

75.27.27 TObject.DispatchStr

Synopsis: Dispatch a string message.

Declaration: `procedure DispatchStr(var message); Virtual`

Visibility: public

Description: `DispatchStr` extracts the message identifier from `Message` and checks the message handler table to see if a handler for the message is found, and calls the handler, passing along the message. If no handler is found, the default `DefaultHandlerStr` (1604) is called.

Errors: None.

See also: `DefaultHandlerStr` (1604), `Dispatch` (1604), `DefaultHandler` (1600)

75.27.28 TObject.GetInterface

Synopsis: Return a reference to an interface.

Declaration: `function GetInterface(const iid: TGuid; out obj) : Boolean`
`function GetInterface(const iidstr: shortstring; out obj) : Boolean`

Visibility: public

Description: `GetInterface` scans the interface tables and returns a reference to the interface `iid`. The reference is stored in `Obj` which should be an interface reference. It returns `True` if the interface was found, `False` if not.

The reference count of the interface is increased by this call.

Errors: If no interface was found, `False` is returned.

See also: `GetInterfaceByStr` (1605)

75.27.29 TObject.GetInterfaceByStr

Synopsis: Return an interface based on its GUID.

Declaration: `function GetInterfaceByStr(const iidstr: shortstring; out obj) : Boolean`

Visibility: public

Description: `GetInterfaceByStr` returns in `obj` a pointer to the interface identified by `iidstr`. The function returns `True` if the interface is indeed implemented by the class, or `False` otherwise.

The `iidstr` is the unique GUID by which the interface was declared.

Errors: The function returns false if the requested interface is not implemented.

See also: `TObject.GetInterfaceEntry` (1606), `TObject.GetInterfaceEntryByStr` (1606)

75.27.30 TObject.GetInterfaceWeak

Synopsis: Get a reference to an interface, not increasing the reference count.

Declaration: `function GetInterfaceWeak(const iid: TGuid; out obj) : Boolean`

Visibility: public

Description: `GetInterfaceWeak` performs the same function as `GetInterface` (1605), but unlike the latter, it will not increase the reference count of the interface.

See also: `TObject.GetInterface` (1605)

75.27.31 TObject.GetInterfaceEntry

Synopsis: Return the interface table entry by GUID.

Declaration: `class function GetInterfaceEntry(const iid: TGuid) : pinterfaceentry`

Visibility: public

Description: `GetInterfaceEntry` returns the internal interface table entry for the interface identified by `iid` (the GUID used in the declaration of the interface). If the interface is not implemented by the class, the function returns `Nil`.

See also: `TObject.GetInterfaceByStr` (1605), `TObject.GetInterfaceEntryByStr` (1606)

75.27.32 TObject.GetInterfaceEntryByStr

Synopsis: Return the interface table entry by string.

Declaration: `class function GetInterfaceEntryByStr(const iidstr: shortstring)
: pinterfaceentry`

Visibility: public

Description: `GetInterfaceEntryByStr` returns the internal interface table entry for the interface identified by `iidstr` (A string representation of the GUID used in the declaration of the interface). If the interface is not implemented by the class, the function returns `Nil`.

See also: `TObject.GetInterfaceByStr` (1605), `TObject.GetInterfaceEntry` (1606)

75.27.33 TObject.GetInterfaceTable

Synopsis: Return a pointer to the table of implemented interfaces for a class.

Declaration: `class function GetInterfaceTable : pinterfacetable`

Visibility: public

Description: `GetInterfaceTable` returns a pointer to the internal table of implemented interfaces for a class. The result will always point to a valid address, if the class implements no interfaces the `EntryCount` field of the interface table will be zero.

See also: `TObject.GetInterfaceByStr` (1605), `TObject.GetInterfaceEntry` (1606)

75.27.34 TObject.UnitName

Synopsis: Unit name.

Declaration: `class function UnitName : ansistring`

Visibility: public

Description: `UnitName` returns the unit name in which the class was defined. The name is obtained from the class definition data the compiler generates for each class.

75.27.35 TObject.QualifiedClassName

Synopsis: Fully qualified classname.

Declaration: `class function QualifiedClassName : ansistring`

Visibility: public

Description: `QualifiedClassName` is the classname `TObject.ClassName` (1601) prepended with the unit name `TObject.UnitName` (1607) of the unit in which the class is defined. As such, this is a unique name for each class in the program.

See also: `TObject.ClassName` (1601), `TObject.UnitName` (1607)

75.27.36 TObject.Equals

Synopsis: Check if two objects are equal.

Declaration: `function Equals(Obj: TObject) : Boolean; Virtual`

Visibility: public

Description: `Equals` returns `True` if the object instance pointer (`Self`) equals the instance pointer `Obj`.
Descendent classes can override to check properties etc. in case the instance pointers are different.

See also: `TObject.GetHashCode` (1607), `TObject.ToString` (1608)

75.27.37 TObject.GetHashCode

Synopsis: Return a hash code for the object.

Declaration: `function GetHashCode : PtrInt; Virtual`

Visibility: public

Description: `GetHashCode` should return a hash code for the object. By default, the numerical (integer) address of `Self` is returned.

Descendent classes can use this to generate better suitable values to be used in a hash table.

See also: `TObject.ToString` (1608), `TObject.Equals` (1607)

75.27.38 TObject.ToString

Synopsis: Return a string representation for the object.

Declaration: `function ToString : ansistring; Virtual`

Visibility: `public`

Description: `ToString` returns by default the class name of the object. It is useful during sending of debug messages.

Descendent classes can override this method to give a better description of the object than just the class name.

See also: `TObject.GetHashCode` ([1607](#)), `TObject.Equals` ([1607](#))

Chapter 76

Reference for unit 'sysutils'

76.1 Used units

Table 76.1: Used units by unit 'sysutils'

Name	Page
BaseUnix	155
errors	760
Linux	996
sysconst	??
System	1335
Unix	2189
unixtype	2229

76.2 Overview

This documentation describes the `sysutils` unit. The `sysutils` unit was started by Gertjan Schouten, and completed by Michael Van Canneyt. It aims to be compatible to the Delphi `sysutils` unit, but in contrast with the latter, it is designed to work on multiple platforms. It is implemented on all supported platforms.

76.3 Type Helpers for basic types.

The `sysutils` unit contains type helpers for basic language types: boolean, ordinals (shortint, smallint, byte, word, integer, cardinal, int64, wqord, nativeint, nativeuint), strings, floating-point and GUID values. They can be used as-is, or they can be descended from to form your own type helpers.

See also: TGuidHelper ([1884](#)), TStringHelper ([1609](#)), TByteHelper ([1851](#)), TShortIntHelper ([1928](#)), TSmallIntHelper ([1955](#)), TWordHelper ([2000](#)), TCardinalHelper ([1857](#)), TIntegerHelper ([1893](#)), TInt64Helper ([1886](#)), TQWordHelper ([1921](#)), TNativeIntHelper ([1908](#)), TBooleanHelper ([1847](#)), TByteBoolHelper ([1849](#)), TWordBoolHelper ([1998](#)), TLongBoolHelper ([1899](#)), TSingleHelper ([1949](#)), TDoubleHelper ([1865](#)), TExtendedHelper ([1878](#))

76.4 Localization support.

Localization support depends on various constants and structures being initialized correctly. On Windows and OS/2 this is done automatically: a widestring manager is installed by default which helps taking care of the current locale when performing various operations on strings. The various internationalization settings (date/time format, currency, language etc) are also initialized correctly on these platforms.

On Unixes, the widestring support is in a separate unit: `cwstring`, which loads the various needed functions from the C library. It should be added manually to the `uses` clause of your program. No internationalization (or localisation) settings are applied by this unit, these must be initialized separately by including the `locale` unit in the `uses` clause of your program.

76.5 Unicode and codepage awareness.

The many functions that deal with filenames in the `sysutils` routines have been changed from `AnsiString` to `RawByteString` so they do not perform implicit codepage conversions to the ANSI code page. At the same time, overloaded versions that accept a Unicode string have been created.

For routines that access actual OS functions using single-byte string APIs, the strings are converted to ensure that the OS routine receives a string with the correct encoding when using single-byte strings. This encoding is normally the `DefaultFileSystemCodePage` (1414) encoding.

On systems with a Unicode I/O API (2-byte strings), the native API is used, meaning that Unicode strings will be passed on as-is, but single-byte strings will be converted (implicitly) to Unicode.

The following is a minimal list of functions that have been changed and duplicated:

Table 76.2:

Name	Description
<code>CreateFile</code> (1694)	Create a new file and return a handle to it.
<code>OpenFile</code> (1700)	Open an existing file and return a file handle.
<code>FileExists</code> (1697)	Check whether a particular file exists in the file system.
<code>DirectoryExists</code> (1680)	Check whether a directory exists in the file system.
<code>FileSetDate</code> (1704)	Set the date of a file.
<code>FileGetAttr</code> (1698)	Return attributes of a file.
<code>FileSetAttr</code> (1704)	Set the attributes of a file.
<code>DeleteFile</code> (1680)	Delete a file from the file system.
<code>RenameFile</code> (1747)	Rename a file.
<code>FileSearch</code> (1702)	Search for a file in a path.
<code>ExeSearch</code> (1687)	Search for an executable.
<code>FindFirst</code> (1706)	Start a file search and return a findhandle.
<code>FindNext</code> (1708)	Find the next entry in a findhandle.
<code>FindClose</code> (1706)	Close a find handle.
<code>FileIsReadOnly</code> (1700)	Check whether a file is read-only.
<code>GetCurrentDir</code> (1727)	Return the current working directory of the application.
<code>SetCurrentDir</code> (1750)	Set the current directory of the application.

The following functions do not interact with the OS, but may nevertheless change the codepage of the strings involved in their operation:

Table 76.3:

Name	Description
ChangeFileExt (1666)	Change the extension of a filename.
ExtractFilePath (1692)	Extract the path from a filename.
ExtractFileDrive (1691)	Extract the drive part from a filename.
ExtractFileName (1691)	Extract the filename part from a full path filename.
ExtractFileExt (1691)	Return the extension from a filename.
ExtractFileDir (1690)	Extract the drive and directory part of a filename.
ExtractShortPathName (1693)	Returns a 8.3 path name.
ExpandFileName (1688)	Expand a relative filename to an absolute filename.
ExpandFileNameCase (1689)	Expand a filename entered as case insensitive to the full path as stored on the disk.
ExtractRelativepath (1692)	Extract a relative path from a filename, given a base directory.
ExpandUNCFileName (1689)	Expand a relative filename to an absolute UNC filename.
IncludeTrailingPathDelimiter (1736)	Add trailing directory separator to a pathname, if needed.
IncludeTrailingBackslash (1736)	Add trailing directory separator to a pathname, if needed.
ExcludeTrailingBackslash (1686)	Strip trailing directory separator from a pathname, if needed.
ExcludeTrailingPathDelimiter (1687)	Strip trailing directory separator from a pathname, if needed.
IncludeLeadingPathDelimiter (1736)	Prepend a path delimiter if there is not already one.
ExcludeLeadingPathDelimiter (1686)	Strip the leading path delimiter of a path.
IsPathDelimiter (1741)	Is the character at the given position a pathdelimiter ?
DoDirSeparators (1682)	Convert known directory separators to the current directory separator.
SetDirSeparators (1751)	Set the directory separators to the known directory separators.
GetDirs (1728)	Return a list of directory names from a path.
ConcatPaths (1671)	Concatenate an array of paths to form a single path.
GetEnvironmentVariable (1729)	Return the value of an environment variable.

76.6 Miscellaneous conversion routines.

Functions for various conversions.

Table 76.4:

Name	Description
BCDToInt (1663)	Convert BCD number to integer
CompareMem (1668)	Compare two memory regions
FloatToStrF (1710)	Convert float to formatted string
FloatToStr (1709)	Convert float to string
FloatToText (1713)	Convert float to string
FormatFloat (1724)	Format a floating point value
GetDirs (1728)	Split string in list of directories
IntToHex (1738)	return hexadecimal representation of integer
IntToStr (1738)	return decimal representation of integer
StrToIntDef (1775)	Convert string to integer with default value
StrToInt (1773)	Convert string to integer
StrToFloat (1772)	Convert string to float
TextToFloat (1780)	Convert null-terminated string to float

76.7 Date/time routines.

Functions for date and time handling.

Table 76.5:

Name	Description
<code>DateTimeToFileDate</code> (1674)	Convert <code>DateTime</code> type to file date
<code>DateTimeToStr</code> (1674)	Construct string representation of <code>DateTime</code>
<code>DateTimeToString</code> (1675)	Construct string representation of <code>DateTime</code>
<code>DateTimeToSystemTime</code> (1676)	Convert <code>DateTime</code> to system time
<code>DateTimeToTimeStamp</code> (1677)	Convert <code>DateTime</code> to timestamp
<code>DateToStr</code> (1677)	Construct string representation of date
<code>Date</code> (1673)	Get current date
<code>DayOfWeek</code> (1678)	Get day of week
<code>DecodeDate</code> (1678)	Decode <code>DateTime</code> to year month and day
<code>DecodeTime</code> (1679)	Decode <code>DateTime</code> to hours, minutes and seconds
<code>EncodeDate</code> (1683)	Encode year, day and month to <code>DateTime</code>
<code>EncodeTime</code> (1684)	Encode hours, minutes and seconds to <code>DateTime</code>
<code>FormatDateTime</code> (1723)	Return string representation of <code>DateTime</code>
<code>IncMonth</code> (1737)	Add 1 to month
<code>IsLeapYear</code> (1740)	Determine if year is leap year
<code>MSecsToTimeStamp</code> (1744)	Convert nr of milliseconds to timestamp
<code>Now</code> (1745)	Get current date and time
<code>StrToDateTime</code> (1770)	Convert string to <code>DateTime</code>
<code>StrToDate</code> (1769)	Convert string to date
<code>StrToTime</code> (1776)	Convert string to time
<code>SystemTimeToDateTime</code> (1780)	Convert system time to datetime
<code>TimeStampToDateTime</code> (1782)	Convert time stamp to <code>DateTime</code>
<code>TimeStampToMSecs</code> (1783)	Convert <code>TimeStamp</code> to number of milliseconds
<code>TimeToStr</code> (1783)	return string representation of <code>Time</code>
<code>Time</code> (1781)	Get current time

76.8 File Name handling routines.

Functions for file manipulation.

Table 76.6:

Name	Description
AnsiCompareFileName (1649)	Compare 2 filenames
AnsiLowerCaseFileName (1653)	Create lowercase filename
AnsiUpperCaseFileName (1661)	Create uppercase filename
AddDisk (1647)	Add disk to list of disk drives
ChangeFileExt (1666)	Change extension of file name
CreateDir (1671)	Create a directory
DeleteFile (1680)	Delete a file
DiskFree (1681)	Free space on disk
DiskSize (1682)	Total size of disk
ExpandFileName (1688)	Create full file name
ExpandFileNameCase (1689)	Create full file name case insensitively
ExpandUNCFileName (1689)	Create full UNC file name
ExtractFileDir (1690)	Extract drive and directory part of filename
ExtractFileDrive (1691)	Extract drive part of filename
ExtractFileExt (1691)	Extract extension part of filename
ExtractFileName (1691)	Extract name part of filename
ExtractFilePath (1692)	Extract path part of filename
ExtractRelativePath (1692)	Construct relative path between two files
FileAge (1693)	Return file age
FileDateToDateTime (1696)	Convert file date to system date
FileExists (1697)	Determine whether a file exists on disk
FileGetAttr (1698)	Get attributes of file
FileGetDate (1699)	Get date of last file modification
FileSearch (1702)	Search for file in path
FileSetAttr (1704)	Get file attributes
FileSetDate (1704)	Get file dates
FindFirst (1706)	Start finding a file
FindNext (1708)	Find next file
GetCurrentDir (1727)	Return current working directory
RemoveDir (1747)	Remove a directory from disk
RenameFile (1747)	Rename a file on disk
SameFileName (1749)	Check whether 2 filenames are the same
SetCurrentDir (1750)	Set current working directory
SetDirSeparators (1751)	Set directory separator characters
FindClose (1706)	Stop searching a file
DoDirSeparators (1682)	Replace directory separator characters

76.9 File input/output routines.

Functions for reading/writing to file.

Table 76.7:

Name	Description
FileCreate (1694)	Create a file and return handle
FileOpen (1700)	Open file end return handle
FileRead (1701)	Read from file
FileSeek (1703)	Set file position
FileTruncate (1705)	Truncate file length
FileWrite (1705)	Write to file
FileClose (1694)	Close file handle

76.10 PChar related functions.

Most PChar functions are the same as their counterparts in the **STRINGS** unit. The following functions are the same :

1. StrCat (1754) : Concatenates two PChar strings.
2. StrComp (1755) : Compares two PChar strings.
3. StrCopy (1755) : Copies a PChar string.
4. StrECopy (1756) : Copies a PChar string and returns a pointer to the terminating null byte.
5. StrEnd (1757) : Returns a pointer to the terminating null byte.
6. StrIComp (1758) : Case insensitive compare of 2 PChar strings.
7. StrLCat (1760) : Appends at most L characters from one PChar to another PChar.
8. StrLComp (1760) : Case sensitive compare of at most L characters of 2 PChar strings.
9. StrLCopy (1761) : Copies at most L characters from one PChar to another.
10. StrLen (1762) : Returns the length (exclusive terminating null byte) of a PChar string.
11. StrLIComp (1763) : Case insensitive compare of at most L characters of 2 PChar strings.
12. StrLower (1763) : Converts a PChar to all lowercase letters.
13. StrMove (1764) : Moves one PChar to another.
14. StrNew (1764) : Makes a copy of a PChar on the heap, and returns a pointer to this copy.
15. StrPos (1766) : Returns the position of one PChar string in another?
16. StrRScan (1767) : returns a pointer to the last occurrence of on PChar string in another one.
17. StrScan (1767) : returns a pointer to the first occurrence of on PChar string in another one.
18. StrUpper (1779) : Converts a PChar to all uppercase letters.

The subsequent functions are different from their counterparts in **STRINGS**, although the same examples can be used.

76.11 Date and time formatting characters.

Various date and time formatting routines accept a format string to format the date and or time. The following characters can be used to control the date and time formatting:

- c** Formats date using `shortdateformat` and formats time using `longtimeformat` if the time is not zero.
- f** Same as **c**, but adds the time even if it is zero.
- d** day of month
- dd** day of month (leading zero)
- ddd** day of week (abbreviation)
- dddd** day of week (full)
- dddddd** `shortdateformat`
- ddddddd** `longdateformat`
- m** month or minutes if preceded by **h** or **hh** specifiers.
- mm** month or minutes if preceded by **h** or **hh** specifiers, with leading zero.
- mmm** month (abbreviation)
- mmmm** month (full)
- y** year (2 digits)
- yy** year (two digits)
- yyyy** year (with century)
- h** hour
- hh** hour (leading zero)
- n** minute
- nn** minute (leading zero)
- s** second
- ss** second (leading zero)
- t** `shorttimeformat`
- tt** `longtimeformat`
- am/pm** use 12 hour clock and display am and pm accordingly
- a/p** use 12 hour clock and display a and p accordingly
- /** insert date separator
- :** insert time separator
- "xx"** literal text
- 'xx'** literal text

z milliseconds

zzz milliseconds(leading zero)

[h] hours including the hours of the full days (i.e. can be > 24).

[hh] hours with leading zero, including the hours of the full days (i.e. can be > 24)

[n] minutes including the minutes of the full hours and days

[nn] minutes with leading zero, including the minutes of the full hours and days

[s] seconds including the seconds of the full minutes, hours and days.

[ss] seconds with leading zero, including the seconds of the full minutes, hours and days.

The forms in square brackets are only allowed if the `fdoInterval` (1633) option is included in the `Options` argument of `FormatDateTime`.

The date and time separators are taken from the `DefaultFormatSettings` (1643) record, unless a `TFormatSettings` (1802) record is passed to the `FormatDateTime` (1723) function.

Note that to include any of the above characters literally in the result string, they must be enclosed in double quotes.

See also: `DefaultFormatSettings` (1643), `TFormatSettings` (1802), `FormatDateTime` (1723), `TFormatDateTimeOption` (1633)

76.12 Formatting strings.

Functions for formatting strings.

Table 76.8:

Name	Description
<code>AdjustLineBreaks</code> (1648)	Convert line breaks to line breaks for system
<code>FormatBuf</code> (1722)	Format a buffer
<code>Format</code> (1715)	Format arguments in string
<code>FmtStr</code> (1714)	Format buffer
<code>QuotedStr</code> (1746)	Quote a string
<code>StrFmt</code> (1757)	Format arguments in a string
<code>StrLFmt</code> (1762)	Format maximum L characters in a string
<code>TrimLeft</code> (1784)	Remove whitespace at the left of a string
<code>TrimRight</code> (1785)	Remove whitespace at the right of a string
<code>Trim</code> (1783)	Remove whitespace at both ends of a string

76.13 String functions.

Functions for handling strings.

Table 76.9:

Name	Description
AnsiCompareStr (1649)	Compare two strings
AnsiCompareText (1650)	Compare two strings, case insensitive
AnsiExtractQuotedStr (1651)	Removes quotes from string
AnsiLastChar (1652)	Get last character of string
AnsiLowerCase (1653)	Convert string to all-lowercase
AnsiQuotedStr (1654)	Quotes a string
AnsiStrComp (1655)	Compare strings case-sensitive
AnsiStrIComp (1655)	Compare strings case-insensitive
AnsiStrLComp (1657)	Compare L characters of strings case sensitive
AnsiStrLIComp (1658)	Compare L characters of strings case insensitive
AnsiStrLastChar (1656)	Get last character of string
AnsiStrLower (1659)	Convert string to all-lowercase
AnsiStrUpper (1660)	Convert string to all-uppercase
AnsiUpperCase (1661)	Convert string to all-uppercase
AppendStr (1662)	Append 2 strings
AssignStr (1663)	Assign value of strings on heap
CompareStr (1668)	Compare two strings case sensitive
CompareText (1669)	Compare two strings case insensitive
DisposeStr (1682)	Remove string from heap
IsValidIdent (1741)	Is string a valid pascal identifier
LastDelimiter (1742)	Last occurrence of character in a string
LeftStr (1742)	Get first N characters of a string
LoadStr (1743)	Load string from resources
LowerCase (1743)	Convert string to all-lowercase
NewStr (1745)	Allocate new string on heap
RightStr (1748)	Get last N characters of a string
StrAlloc (1753)	Allocate memory for string
StrBufSize (1753)	Reserve memory for a string
StrDispose (1756)	Remove string from heap
StrPas (1765)	Convert PChar to pascal string
StrPCopy (1766)	Copy pascal string
StrPLCopy (1766)	Copy N bytes of pascal string
UpperCase (1795)	Convert string to all-uppercase

76.14 Constants, types and variables

76.14.1 Constants

`ConfigExtension : string = '.cfg'`

`ConfigExtension` is the default extension used by the `GetAppConfigFile (1726)` call. It can be set to any valid extension for the current OS.

`CPUEndian = TEndian.Little`

`CPUEndian` describes whether the current CPU is little or big endian.

`DateDelta = 693594`

Days between 1/1/0001 and 12/31/1899.

`DriveDelim = DriveSeparator`

`DriveDelim` refers to the system unit's `DriveSeparator` constant, it is for Delphi compatibility only.

`EmptyStr : string = ''`

Empty String Constant.

`EmptyWideStr : WideString = ''`

Empty wide string.

`faAnyFile = $000001FF`

Use this attribute in the `FindFirst` (1706) call to find all matching files.

`faArchive = $00000020`

Attribute of a file, meaning the file has the archive bit set. Used in `TSearchRec` (1637) and `FindFirst` (1706)

`faCompressed = $00000800platform`

`faTemporary` can be returned by `FindFirst` (1706) or `FindNext` (1708) to indicate that a returned file is compressed (on file systems that support this).

`faDirectory = $00000010`

Attribute of a file, meaning the file is a directory. Used in `TSearchRec` (1637) and `FindFirst` (1706)

`faEncrypted = $00004000platform`

`faEncrypted` can be returned by `FindFirst` (1706) or `FindNext` (1708) to indicate that a returned file is encrypted (on file systems that support this).

`faHidden = $00000002platform`

Attribute of a file, meaning the file is read-only. Used in `TSearchRec` (1637) and `FindFirst` (1706)

`faNormal = $00000080`

`faNormal` can be used in `FindFirst` (1706) to indicate that normal files must be included in the result.

`faReadOnly = $00000001`

Attribute of a file, meaning the file is read-only. Used in `TSearchRec` (1637) and `FindFirst` (1706)

`faSymLink = $00000400platform`

`faSymLink` means the file (as returned e.g. by `FindFirst (1706)`/`FindNext (1708)`), is a symlink. It's ignored under Windows.

`faSysFile = $00000004platform`

Attribute of a file, meaning the file is a system file. Used in `TSearchRec (1637)` and `FindFirst (1706)`

`faTemporary = $00000100platform`

`faTemporary` can be returned by `FindFirst (1706)` or `FindNext (1708)` to indicate that a returned file is a temporary file.

`faVirtual = $00010000platform`

`faVirtual` can be returned by `FindFirst (1706)` or `FindNext (1708)` to indicate that a returned file is virtual (on file systems that support this).

`faVolumeId = $00000008deprecated; platform`

Attribute of a file, meaning the entry contains the volume ID. Used in `TSearchRec (1637)` and `FindFirst (1706)`

`feInvalidHandle = THandle(- 1)`

`feInvalidHandle` is the return value of `FileOpen (1700)` in case of an error.

`fmOpenRead = $0000`

`fmOpenRead` is used in the `FileOpen (1700)` call to open a file in read-only mode.

`fmOpenReadWrite = $0002`

`fmOpenReadWrite` is used in the `FileOpen (1700)` call to open a file in read-write mode.

`fmOpenWrite = $0001`

`fmOpenWrite` is used in the `FileOpen (1700)` call to open a file in write-only mode.

`fmShareCompat = $0000`

`fmOpenShareCompat` is used in the `FileOpen (1700)` call OR-ed together with one of `fmOpenReadWrite (1619)`, `fmOpenRead (1619)` or `fmOpenWrite (1619)`, to open a file in a sharing modus that is equivalent to sharing implemented in MS-DOS.

`fmShareDenyNone = $0040`

`fmShareDenyNone` is used in the `FileOpen (1700)` call OR-ed together with one of `fmOpenReadWrite (1619)`, `fmOpenRead (1619)` or `fmOpenWrite (1619)`, to open a file so other processes can read/write the file as well.

```
fmShareDenyRead = $0030
```

`fmOpenShareRead` is used in the `FileOpen` (1700) call OR-ed together with one of `fmOpenReadWrite` (1619), `fmOpenRead` (1619) or `fmOpenWrite` (1619), to open a file so other processes cannot read from it.

This constant only works on Windows, because other operating systems do not support this constants.

```
fmShareDenyWrite = $0020
```

`fmOpenShareDenyWrite` is used in the `FileOpen` (1700) call OR-ed together with one of `fmOpenReadWrite` (1619), `fmOpenRead` (1619) or `fmOpenWrite` (1619), to open a file so other processes cannot write to it, they can only read.

```
fmShareExclusive = $0010
```

`fmOpenShareExclusive` is used in the `FileOpen` (1700) call OR-ed together with one of `fmOpenReadWrite` (1619), `fmOpenRead` (1619) or `fmOpenWrite` (1619), to open a file exclusively.

```
fmShareNoLocking = $0080
```

```
fsFromBeginning = 0
```

`fsFromBeginning` is used to indicate in the `FileSeek` (1703) call that a seek operation should be started at the start of the file.

```
fsFromCurrent = 1
```

`fsFromBeginning` is used to indicate in the `FileSeek` (1703) call that a seek operation should be started at the current position in the file.

```
fsFromEnd = 2
```

`fsFromBeginning` is used to indicate in the `FileSeek` (1703) call that a seek operation should be started at the last position in the file.

```
GUID_NULL : TGuid = '{00000000-0000-0000-0000-000000000000}'
```

NULL GUID constant.

```
HexDisplayPrefix : string = '$'
```

`HexDisplayPrefix` is used by the formatting routines to indicate that the number which follows the prefix is in Hexadecimal notation.

```
HoursPerDay = 24
```

Number of hours in a day.

```
INVALID_HANDLE_VALUE = DWORD(- 1)
```

JulianEpoch = TDateTime(- 2415018.5)

Starting point of the Julian calendar.

LeadBytes : Set of AnsiChar = []

LeadBytes contains the set of bytes that serve as lead byte in a MBCS string.

MaxCurrency : Currency = 922337203685477.5807

Maximum currency value.

MaxDateTime : TDateTime = 2958465.999999999

Maximum TDateTime value.

MAX_PATH = MaxPathLen

MAX_PATH is the maximum number of characters that a filename (including path) can contain on the current operating system.

MinCurrency : Currency = - 922337203685477.5808

Minimum Currency value.

MinDateTime : TDateTime = - 693593.999999999

Minimum TDateTime value.

MinsPerDay = HoursPerDay * MinsPerHour

Number of minutes per day.

MinsPerHour = 60

Number of minutes per hour.

MonthDays : Array[Boolean] of TDayTable = ((31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31), (31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31))

Array with number of days in the months for leap and non-leap years.

MSecsPerDay = SecsPerDay * MSecsPerSec

Number of milliseconds per day.

MSecsPerSec = 1000

Number of milliseconds per second.

`NullStr : PString = @ EmptyStr`

Pointer to an empty string.

`PathDelim = DirectorySeparator`

`PathDelim` refers to the system unit's `DirectorySeparator` constant, it is for Delphi compatibility only.

`PathSep = PathSeparator`

`PathSep` refers to the system unit's `PathSeparator` constant, it is for Delphi compatibility only.

`pfBCB4Produced = $08000000`

Not used in Free Pascal.

`pfDelphi4Produced = $0C000000`

Not used in Free Pascal.

`pfDesignOnly = $00000002`

Package is a design-time only package.

`pfExeModule = $00000000`

Package is an executable.

`pfIgnoreDupUnits = $00000008`

Ignore duplicate units in package.

`pfLibraryModule = $80000000`

Package is a library.

`pfModuleTypeMask = $C0000000`

Mask for module type flags.

`pfNeverBuild = $00000001`

Never-build flag was specified when compiling package.

`pfPackageModule = $40000000`

Package is a real package (not exe).

`pfProducerMask = $0C000000`

Mask for producer flags.

`pfProducerUndefined = $04000000`

Not used in Free Pascal.

`pfRunOnly = $00000004`

Package is a run-time only package.

`pfV3Produced = $00000000`

Not used in Free Pascal.

`RTL_SIGBUS = 4`

Bus error signal number (Unix only).

`RTL_SIGDEFAULT = - 1`

Default signal handler (Unix only).

`RTL_SIGFPE = 1`

Floating Point Error signal number (Unix only).

`RTL_SIGILL = 3`

Illegal instruction signal number (Unix only).

`RTL_SIGINT = 0`

INTERRUPT signal number (Unix only).

`RTL_SIGLAST = RTL_SIGQUIT`

Last signal number (Unix only).

`RTL_SIGQUIT = 5`

QUIT signal number (Unix only).

`RTL_SIGSEGV = 2`

Segmentation fault signal number (Unix only).

`SecsPerDay = MinsPerDay * SecsPerMin`

Number of seconds per day.

`SecsPerHour = SecsPerMin * MinsPerHour`

`SecsPerHour` is a constant indicating the number of seconds in an hour (3600).

```
SecsPerMin = 60
```

Number of seconds per minute.

```
SwitchChars = ['-']
```

The characters in this set will be used by the `FindCmdLineSwitch` (1706) function to determine whether a command-line argument is a switch (an option) or a value. If the first character of an argument is in `SwitchChars`, it will be considered an option or switch.

```
SysConfigDir : string = ''
```

`SysConfigDir` is the default system configuration directory. It is set at application startup by the **sysutils** initialization routines.

This directory may be returned by the `GetAppConfigDir` (1726) call on some systems.

```
ufImplicitUnit = $10
```

Unit was implicitly imported into package (did not appear in package contains list).

```
ufMainUnit = $01
```

Unit is the main unit of the package.

```
ufOrgWeakUnit = $08
```

Unit is the original weak packaged unit.

```
ufPackageUnit = $02
```

Unit is a packaged unit (appeared in package contains list).

```
ufWeakPackageUnit = ufPackageUnit or ufWeakUnit
```

Weak (original or not) packaged unit.

```
ufWeakUnit = $04
```

Unit is a weak packaged unit.

```
UnixDateDelta = Trunc(UnixEpoch)
```

Number of days between 1.1.1900 and 1.1.1970.

```
UnixEpoch = JulianEpoch + TDateTime(2440587.5)
```

Starting point of the unix calendar (1/1/1970).

76.14.2 Types

`EHeapException = EHeapMemoryError`

`EHeapMemoryError` is raised when an error occurs in the heap management routines.

`ExceptClass = Class of Exception`

`ExceptClass` is a `Exception` ([1821](#)) class reference.

```

Int128Rec = packed record
case Integer of
0: (
public
  Lo : QWord
  ;
  Hi : QWord;
);
1: (
public
  DWords : Array[0..3] of DWord;
)
  ;
2: (
public
  Words : Array[0..7] of Word;
);
3: (
public
  Bytes
    : Array[0..15] of Byte;
);
end

```

`Int128Rec` is a record defining a 128-bit integer. It is made up of 2 QWords or 4 DWords or 8 words or 16 bytes.

```

Int64Rec = packed record
case Integer of
0: (
public
  Lo : Cardinal
  ;
  Hi : Cardinal;
);
1: (
public
  Words : Array[0..3] of Word;
);
2: (
public
  Bytes : Array[0..7] of Byte;
);
end

```

Int64Rec can be used to extract the parts of a Int64: the high and low cardinal, or a zero-based array of 4 words, or a zero based array of 8 bytes. Note that the meaning of the High and Low parts are different on various CPUs.

```
LongRec = packed record
case Integer of
0: (
public
  Lo : Word;
  Hi : Word;
);
1: (
public
  Bytes : Array[0..3] of Byte;
);
end
```

LongRec can be used to extract the parts of an long Integer: the high and low word, or the 4 separate bytes as a zero-based array of bytes. Note that the meaning (or ordering) of High and Low parts are different on various CPUs, and may differ from what is shown in the declaration: the ordering depends on the endianness of the CPU.

```
nibble = 0..15
```

Half byte type.

```
OWordRec = packed record
case Integer of
0: (
public
  Lo : QWord
  ;
  Hi : QWord;
);
1: (
public
  DWords : Array[0..3] of DWord;
)
;
2: (
public
  Words : Array[0..7] of Word;
);
3: (
public
  Bytes
  : Array[0..15] of Byte;
);
end
```

OWordRec is a record defining a 128-bit integer. It is made up of 2 QWords or 4 DWords or 8 words or 16 bytes.

```
PByteArray = ^TByteArray
```

Generic pointer to TByteArray ([1627](#)). Use to access memory regions as a byte array.

```
PDayTable = ^TDayTable
```

Pointer to TDayTable type.

```
PString = ObjPas.PString
```

Pointer to a ansistring.

```
PSysCharSet = ^TSysCharSet
```

Pointer to TSysCharSet ([1639](#)) type.

```
PWordarray = ^TWordArray
```

Generic pointer to TWordArray ([1641](#)). Use to access memory regions as a word array.

```
TAnsiCharArray = Array of AnsiChar
```

```
TAnsiStringArray = Array of Ansistring
```

```
TArchitectures = Set of TOSVersion.TArchitecture
```

```
TBeepHandler = procedure
```

TBeepHandler is the prototype used by the OnBeep ([1644](#)) handler. This in turn is called by the Beep ([1664](#)) call to actually implement the beep functionality. The call takes no arguments.

```
TByteArray = Array[0..32767] of Byte
```

TByteArray is a generic array definition, mostly for use as a base type of the PByteArray ([1627](#)) type.

```
TByteBitIndex = 0..7
```

TByteBitIndex is the range of allowed bit indexes for a 8-bit unsigned integer type, it is used in the 8-bit helper TByteHelper ([1851](#)).

```
TByteNibbleIndex = 0..1
```

```
TByteOverlay = bitpacked record
case Integer of
0: (
public
  AsBit
```

```

    : Array[TByteBitIndex] of Boolean;
);
1: (
public
    AsNibble : Array
        [TByteNibbleIndex] of nibble;
);
2: (
public
    AsByte : Byte;
);
end

```

Helper structure for quick data extraction from a byte-typed ordinal value.

TBytes = Array of Byte

TBytes defines a dynamic array of bytes. This can be used to typecast e.g. strings to manipulate them byte for byte.

TCardinalBitIndex = 0..31

TIntegerBitIndex is the range of allowed bit indexes for a 32-bit unsigned integer type, it is used in the 32-bit helper TCardinalHelper (1857).

TCardinalByteIndex = TDwordByteIndex

TCardinalNibbleIndex = TDwordNibbleIndex

TCardinalOverlay = TDwordOverlay

Helper structure for quick data extraction from a cardinal-typed ordinal value.

TCardinalWordIndex = TDwordWordIndex

TCharArray = Array of AnsiChar

TCharArray is a dynamic array of characters.

TCmdLineSwitchType = (clstValueNextParam, clstValueAppended)

Table 76.10: Enumeration values for type TCmdLineSwitchType

Value	Explanation
clstValueAppended	
clstValueNextParam	

TCmdLineSwitchTypes = Set of TCmdLineSwitchType

TCompareOption = system.TCompareOption

TCompareOption is an alias for System.TCompareOption (1385).

TCompareOptions = system.TCompareOptions

TCompareOptions is an alias for System.TCompareOptions (1386).

TCreateGUIDFunc = function(out GUID: TGuid) : Integer

TCreateGUIDFunc is the prototype for a GUID creation handler. On return, the GUID argument should contain a new (unique) GUID. The return value of the function should be zero for success, nonzero for failure.

TDayTable = Array[1..12] of Word

Array of day names.

TDelphiPlatforms = Set of TOSVersion.TDelphiPlatform

TDwordBitIndex = TCardinalBitIndex

TDwordByteIndex = 0..3

TDwordNibbleIndex = 0..7

```
TDwordOverlay = bitpacked record
case Integer of
0: (
public
  AsBit
    : Array[TDwordBitIndex] of Boolean;
);
1: (
public
  AsNibble : Array
    [TDwordNibbleIndex] of nibble;
);
2: (
public
  AsByte : Array[TDwordByteIndex
    ] of Byte;
);
3: (
public
  AsWord : Array[TDwordWordIndex] of Word
```

```

    ;
  );
4: (
public
  AsDword : DWord;
);
5: (
public
  AsByteOverlay
    : Array[TDwordByteIndex] of TByteOverlay;
);
6: (
public
  AsWordOverlay
    : Array[TDwordWordIndex] of TWordOverlay;
);
end

```

Helper structure for quick data extraction from a dword-typed ordinal value.

```
TDwordWordIndex = 0..1
```

```
TEndian = ObjPas.TEndian
```

TEndian indicates whether the platform is little endian (LSB first), or big endian (MSB first).

```
TEventType = (etCustom, etInfo, etWarning, etError, etDebug)
```

Table 76.11: Enumeration values for type TEventType

Value	Explanation
etCustom	Custom log event, with application-specific meaning.
etDebug	Debug message.
etError	Error condition message.
etInfo	General information event message.
etWarning	Warning message.

TEventType is a type to be used by logging mechanisms (in particular, the TCustomApplication and TEventLog classes. It can be used to filter events, and write only certain types of event to the event log.

```
TEventTypes = Set of TEventType
```

TEventTypes is a set type of TEventType, defined for convenience. It is used in the custom application classes for logging purposes.

```
TExecuteFlags= Set of (ExecInheritsHandles)
```

Table 76.12: Enumeration values for type

Value	Explanation
<code>ExecInheritsHandles</code>	The new process inherits all (file) handles owned by the current process.

`TExecuteFlags` is a set of flags to influence the behaviour of the `ExecuteProcess` (1687) call.

`TFileName = string`

`TFileName` is used in the `TSearchRec` (1637) definition.

`TFilenameCaseMatch = (mkNone, mkExactMatch, mkSingleMatch, mkAmbiguous)`

Table 76.13: Enumeration values for type `TFilenameCaseMatch`

Value	Explanation
<code>mkAmbiguous</code>	More than one file will match the filename in a case-insensitive way.
<code>mkExactMatch</code>	The filename can be used to refer to a file on the system (findfirst will find it).
<code>mkNone</code>	No file was found.
<code>mkSingleMatch</code>	Exactly one match was found, but case didn't match.

`TFilenameCaseMatch` describes how `ExpandFileNameCase` (1689) found the file.

`TFilenameCaseMatch = (mkNone, mkExactMatch, mkSingleMatch, mkAmbiguous)`

Table 76.14: Enumeration values for type `TFilenameCaseMatch`

Value	Explanation
<code>mkAmbiguous</code>	More than one file will match the filename in a case-insensitive way.
<code>mkExactMatch</code>	The filename can be used to refer to a file on the system (findfirst will find it).
<code>mkNone</code>	No file was found.
<code>mkSingleMatch</code>	Exactly one match was found, but case didn't match.

`TFilenameCaseMatch` describes how `ExpandFileNameCase` (1689) found the file.

`TFileRec = FileRec`

Alias for `FileRec` (1609) for Delphi compatibility.

`TFileSearchOption = (sfoImplicitCurrentDir, sfoStripQuotes)`

Table 76.15: Enumeration values for type TFileSearchOption

Value	Explanation
sfoImplicitCurrentDir	Always search the current directory first, even if it is not specified.
sfoStripQuotes	Strip quotes from the components in the search path.

TFileSearchOption enumerates the options that can be used in the FileSearch call to control the behaviour of the search mechanism

TFileSearchOptions = Set of TFileSearchOption

TFileSearchOptions is a set of TFileSearchOption (1631) values, used in the FileSearch (1702) call when searching for files.

TFloatFormat = (ffGeneral, ffExponent, ffFixed, ffNumber, ffCurrency)

Table 76.16: Enumeration values for type TFloatFormat

Value	Explanation
ffCurrency	Monetary format.
ffExponent	Scientific format.
ffFixed	Fixed point format.
ffGeneral	General number format.
ffNumber	Fixed point format with thousand separator.

TFloatFormat is used to determine how a float value should be formatted in the FloatToText (1713) function.

```
TFloatRec = record
public
    Exponent : Integer;
    Negative : Boolean
;
    Digits : Array[0..18] of AnsiChar;
end
```

TFloatRec is used to describe a floating point value by the FloatToDecimal (1709) function.

```
TFloatValue = (fvExtended, fvCurrency, fvSingle, fvReal, fvDouble, fvComp
)
```

Table 76.17: Enumeration values for type TFloatValue

Value	Explanation
fvComp	Comp value.
fvCurrency	Currency value.
fvDouble	Double value.
fvExtended	Extended value.
fvReal	Real value.
fvSingle	Single value.

TFloatValue determines which kind of value should be returned in the (untyped) buffer used by the TextToFloat (1780) function.

```
TFormatDateTimeOption = (fdoInterval)
```

Table 76.18: Enumeration values for type TFormatDateTimeOption

Value	Explanation
fdoInterval	Format the time as an interval, 24+hours are presented as such.

TFormatDateTimeOption enumerates possible options to the FormatDateTime (1723) routine.

```
TFormatDateTimeOptions = Set of TFormatDateTimeOption
```

TFormatDateTimeOptions is a set of TFormatDateTimeOptions (1633), and is used in the last argument of FormatDateTime (1723)

```
TGetAppNameEvent = function : string
```

This callback type is used by the OnGetApplicationName (1645) to return an alternative application name.

```
TGetTempDirEvent = function(Global: Boolean) : string
```

Function prototype for OnGetTempDir (1645) handler.

```
TGetTempFileEvent = function(const Dir: string; const Prefix: string
)
: string
```

Function prototype for OnGetTempFile (1645) handler.

```
TGetVendorNameEvent = function : string
```

TGetVendorNameEvent is the function prototype for the OnGetVendorName (1645) callback, used by the VendorName (1796) function.

```
THandle = System.THandle
```

`THandle` refers to the definition of `THandle` in the system unit, and is provided for backward compatibility only.

```
TInitFormatSettingsHook = procedure(const aLocale: string;
  var aSettings: TFormatSettings)
```

```
TInt64BitIndex = 0..63
```

`TInt64BitIndex` is the range of allowed bit indexes for a 64-bit integer type, it is used in the 64-bit helper `TInt64Helper` (1886).

```
TInt64ByteIndex = TQwordByteIndex
```

```
TInt64DwordIndex = TQwordDwordIndex
```

```
TInt64NibbleIndex = TQwordNibbleIndex
```

```
TInt64Overlay = TQwordOverlay
```

Helper structure for quick data extraction from a `int64`-typed ordinal value.

```
TInt64WordIndex = TQwordWordIndex
```

```
TIntegerSet = Set of
```

`TIntegerSet` is a generic integer subrange set definition whose size fits in a single integer.

```
TLocaleOptions = (loInvariantLocale, loUserLocale)
```

Table 76.19: Enumeration values for type `TLocaleOptions`

Value	Explanation
<code>loInvariantLocale</code>	Use system locale.
<code>loUserLocale</code>	Use user locale.

`TLocaleOptions` is used to indicate what locale should be used for some string operations such as `UpperCase` (1795), `LowerCase` (1743), `CompareText` (1669), `SameText` (1750). The following values are possible:

`loInvariantLocale` Use system locale.

`loUserLocale` Use user locale.

For consistent behaviour independent of the user or service applications use `loInvariantLocale`.

```
TLongIntBitIndex = 0..31
```

TLongIntBitIndex is the range of allowed bit indexes for a 32-bit signed integer type, it is used in the 32-bit helper TLongintHelper ([1901](#)).

TLongIntByteIndex = TDwordByteIndex

TLongIntNibbleIndex = TDwordNibbleIndex

TLongIntOverlay = TDwordOverlay

TLongIntWordIndex = TDwordWordIndex

TMbcsByteType = (mbSingleByte, mbLeadByte, mbTrailByte)

Table 76.20: Enumeration values for type TMbcsByteType

Value	Explanation
mbLeadByte	Uses lead-byte.
mbSingleByte	Single bytes.
mbTrailByte	Uses trailing byte.

Type of multi-byte character set.

TMonthNameArray = Array[1..12] of string

TMonthNameArray is used in the month long and short name arrays.

TNativeIntBitIndex = TInt64BitIndex

TNativeUIntBitIndex = TQwordBitIndex

TNibbleBitIndex = 0..3

TPlatforms = Set of TOSVersion.TPlatform

TProcedure = procedure

TProcedure is a general definition of a procedural callback.

TQwordBitIndex = 0..63

TQwordBitIndex is the range of allowed bit indexes for a 64-bit unsigned integer type, it is used in the 64-bit helper TQWordHelper ([1921](#)).

```
TQwordByteIndex = 0..7
```

```
TQwordDwordIndex = 0..1
```

```
TQwordNibbleIndex = 0..15
```

```
TQwordOverlay = bitpacked record
case Integer of
0: (
public
  AsBit
    : Array[TQwordBitIndex] of Boolean;
);
1: (
public
  AsNibble : Array
    [TQwordNibbleIndex] of nibble;
);
2: (
public
  AsByte : Array[TQwordByteIndex
    ] of Byte;
);
3: (
public
  AsWord : Array[TQwordWordIndex] of Word
    ;
);
4: (
public
  AsDword : Array[TQwordDwordIndex] of DWord;
);
5: (
public
  AsQword : QWord;
);
6: (
public
  AsByteOverlay : Array
    [TQwordByteIndex] of TByteOverlay;
);
7: (
public
  AsWordOverlay
    : Array[TQwordWordIndex] of TWordOverlay;
);
8: (
public
  AsDwordOverlay
    : Array[TQwordDwordIndex] of TDwordOverlay;
);
end
```

Helper structure for quick data extraction from a qword-typed ordinal value.

```
TQwordWordIndex = 0..3
```

```
TReplaceFlags= Set of (rfReplaceAll, rfIgnoreCase)
```

Table 76.21: Enumeration values for type

Value	Explanation
rfIgnoreCase	Search case insensitive.
rfReplaceAll	Replace all occurrences of the search string with the replacement string.

TReplaceFlags determines the behaviour of the StringReplace (1759) function.

```
TSearchRec = TRawbyteSearchRec
```

TSearchRec is a search handle description record. It is initialized by a call to FindFirst (1706) and can be used to do subsequent calls to FindNext (1708). It contains the result of these function calls. It must be used to close the search sequence with a call to FindClose (1706).

Remark Not all fields of this record should be used. Some of the fields are for internal use only. (PathOnly for example, is only provided for Kylix compatibility)

```
TShortIntBitIndex = 0..7
```

TShortIntBitIndex is the range of allowed bit indexes for a 8-bit signed integer type, it is used in the 8-bit helper TShortIntHelper (1928).

```
TShortIntNibbleIndex = TByteNibbleIndex
```

```
TShortIntOverlay = TByteOverlay
```

Helper structure for quick data extraction from a shortint-typed ordinal value.

```
TShortStringArray = Array of ShortString
```

```
TSignalState = (ssNotHooked, ssHooked, ssOverridden)
```

Table 76.22: Enumeration values for type TSignalState

Value	Explanation
ssHooked	A signal handler is set by the RTL code for the signal.
ssNotHooked	No signal handler is set for the signal.
ssOverridden	A signal handler was set for the signal by third-party code.

TSignalState indicates the state of a signal handler in a unix system for a particular signal.

```
TSmallIntBitIndex = 0..15
```

TSmallIntBitIndex is the range of allowed bit indexes for a 16-bit signed integer type, it is used in the 16-bit helper TSmallIntHelper (1955).

```
TSmallIntByteIndex = TWordByteIndex
```

```
TSmallIntNibbleIndex = TWordNibbleIndex
```

```
TSmallIntOverlay = TWordOverlay
```

Helper structure for quick data extraction from a smallint-typed ordinal value.

```
TStringArray = Array of string
```

TStringArray is a dynamic array of strings.

```
TStringBuilder = TUNICODESTRINGBUILDER
```

```
TStringSplitOptions = (None, ExcludeEmpty, ExcludeLastEmpty)
```

Table 76.23: Enumeration values for type TStringSplitOptions

Value	Explanation
ExcludeEmpty	Do not include empty strings in the result.
ExcludeLastEmpty	Only exclude the last element if it is empty.
None	No options specified. The default.

TStringSplitOptions describes the possible options for TStringHelper.Split (1609).

```
TSymLinkRec = TRawbyteSymLinkRec
```

TSymLinkRec contains information about the target of a symlink. Its actual type depends on the OS file API: it is either TRawbyteSymLinkRec (1808) or TUnicodeSymLinkRec (1810). Both types are used in the FileGetSymLinkTarget (1700) call.

TargetName #ShortDescr:

Size #ShortDescr:

Attr #ShortDescr:

Mode #ShortDescr:

TimeStamp #ShortDescr:

```
TSysCharSet = Set of AnsiChar
```

Generic set of characters type.

```
TSysLocale = record
public
  DefaultLCID : Integer;
  PriLangID :
    Integer;
  SubLangID : Integer;
case Byte of
1: (
public
  FarEast
    : Boolean;
  MiddleEast : Boolean;
);
2: (
public
  MBCS : Boolean
    ;
  RightToLeft : Boolean;
);
end
```

TSysLocale describes the current locale. If Fareast or MBCS is True, then the current locale uses a Multi-Byte Character Set. If MiddleEast or RightToLeft is True then words and sentences are read from right to left.

```
TSystemTime = record
case Integer of
0: (
public
  Year : Word;
  Month : Word;
  Day : Word;
  DayOfWeek : Word;
  Hour : Word;
  Minute : Word;
  Second : Word;
  MilliSecond : Word;
);
1: (
public
  wYear : Word;
  wMonth : Word;
  wDay : Word;
  wDayOfWeek : Word
    ;
  wHour : Word;
  wMinute : Word;
  wSecond : Word;
  wMilliseconds
    : Word;
);
end
```


The System time structure contains the date/time in a human-understandable format.

```
TTerminateProc = function : Boolean
```

TTerminateProc is the procedural type which should be used when adding exit procedures.

```
TTextRec = TextRec
```

Alias for TextRec ([1609](#)) for Delphi compatibility.

```
TTimeStamp = record
public
    Time : LongInt;
    Date : LongInt;
end
```

TTimeStamp contains a timestamp, with the date and time parts specified as separate longint values: the number of milliseconds since midnight, and the number of days since 1/1/0001 (plus one).

```
TUnicodeCharArray = Array of UnicodeChar
```

TUnicodeCharArray is a definition of an array of UnicodeChar elements.

```
TUnicodeStringArray = Array of UnicodeString
```

```
TUseBoolStrs = (False, True)
```

Table 76.24: Enumeration values for type TUseBoolStrs

Value	Explanation
False	Numerical values should be used (0 and 1).
True	String representations should be used.

TUseBoolStrs is an enumerated type which indicates whether bool strs must be used when converting boolean to strings.

Because the identifiers True and False are already used for the system boolean type, You must use the scoped identifiers to get the TUseBoolStrs versions of these 2 identifiers. Failure to do so will result in an error.

```
TUTF8StringDynArray = Array of UTF8String
```

```
TWeekNameArray = Array[1..7] of string
```

TWeekNameArray is used in the day long and short name arrays.

```
TWideCharArray = Array of WideChar
```

`TWideStringArray = Array of WideString`

`TWordArray = Array[0..16383] of Word`

`TWordArray` is a generic array definition, mostly for use as a base type of the `PWordArray` (1627) type.

`TWordBitIndex = 0..15`

`TIntegerBitIndex` is the range of allowed bit indexes for a 16-bit unsigned integer type, it is used in the 16-bit helper `TWordHelper` (2000).

`TWordByteIndex = 0..1`

`TWordNibbleIndex = 0..3`

```
TWordOverlay = bitpacked record
case Integer of
0: (
public
  AsBit
    : Array[TWordBitIndex] of Boolean;
);
1: (
public
  AsNibble : Array
    [TWordNibbleIndex] of nibble;
);
2: (
public
  AsByte : Array[TWordByteIndex
    ] of Byte;
);
3: (
public
  AsWord : Word;
);
4: (
public
  AsByteOverlay
    : Array[TWordByteIndex] of TByteOverlay;
);
end
```

Helper structure for quick data extraction from a word-typed ordinal value.

```
WordRec = packed record
public
  Lo : Byte;
  Hi : Byte;
end
```

`LongRec` can be used to extract the parts of a word: the high and low byte. Note that the meaning of the High and Low parts are different on various CPUs.

76.14.3 Variables

`CurrencyDecimals` : `Bytdeprecated`

`CurrencyDecimals` is the number of decimals to be used when formatting a currency. It is used by the float formatting routines. The initialization routines of the `SysUtils` unit initialize this string with a value conforming to the regional preferences of the user or system regional default.

This constant is deprecated. You should use `DefaultFormatSettings.CurrencyFormat` (1643) instead.

`CurrencyFormat` : `Bytdeprecated`

`CurrencyFormat` is the default format string for positive currencies. It is used by the float formatting routines. The initialization routines of the `SysUtils` unit initialize this string with a value conforming to the regional preferences of the user or system regional default.

This constant is deprecated. You should use `DefaultFormatSettings.CurrencyFormat` (1643) instead.

`CurrencyString` : `stringdeprecated`

`CurrencyString` is the currency symbol for the current locale. It is used by the float formatting routines. The initialization routines of the `SysUtils` unit initialize this string with a value conforming to the regional preferences of the user or system regional default.

This constant is deprecated. You should use `DefaultFormatSettings.CurrencyString` (1643) instead.

`DateSeparator` : `AnsiChardeprecated`

`DateSeparator` is the character used by various date/time conversion routines as the character that separates the day from the month and the month from the year in a date notation. It is used by the date formatting routines. The initialization routines of the `SysUtils` unit initialize this string with a value conforming to the regional preferences of the user or system regional default.

This constant is deprecated. You should use `DefaultFormatSettings.DateSeparator` (1643) instead.

`DecimalSeparator` : `AnsiChardeprecated`

`DecimalSeparator` is used to display the decimal symbol in floating point numbers or currencies. It is used by the float formatting routines. The initialization routines of the `SysUtils` unit initialize this string with a value conforming to the regional preferences of the user or system regional default.

This constant is deprecated. You should use `DefaultFormatSettings.DecimalSeparator` (1643) instead.

```
DefaultFormatSettings : TFormatSettings = (CurrencyFormat: 1; NegCurrFormat
    : 5; ThousandSeparator: ','; DecimalSeparator: '.'; CurrencyDecimals
    : 2; DateSeparator: '-'; TimeSeparator: ':'; ListSeparator: ','; CurrencyString
    : '$'; ShortDateFormat: 'd/m/y'; LongDateFormat: 'dd" "mmmm" "yyyy'
    ; TimeAMString: 'AM'; TimePMString: 'PM'; ShortTimeFormat: 'hh:nn'
    ; LongTimeFormat: 'hh:nn:ss'; ShortMonthNames: ('Jan', 'Feb', 'Mar'
    , 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec');
    LongMonthNames: ('January', 'February', 'March', 'April', 'May',
    'June', 'July', 'August', 'September', 'October', 'November', 'December'
    ); ShortDayNames: ('Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat'
    ); LongDayNames: ('Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday'
    , 'Friday', 'Saturday'); TwoDigitYearCenturyWindow: 50)
```

`DefaultFormatSettings` contains the default settings for all type of formatting constants. If no thread-specific values are specified when a formatting function is called, this record is used as a default.

All other formatting constants refer to the fields of this variable using absolute addressing.

`FalseBoolStrs` : Array of string

`FalseBoolStrs` contains the strings that will result in a `False` return value by `StrToBool` (1768).

`FormatSettings` : `TFormatSettings`

`FormatSettings` is provided for Delphi compatibility, and refers to the `DefaultFormatSettings` (1643) variable.

`InitFormatSettingsHook` : `TInitFormatSettingsHook`

`ListSeparator` : `AnsiChar` deprecated

`ListSeparator` is the character used in lists of values. It is locale dependent.

`LongDateFormat` : string deprecated

`LongDateFormat` contains a template to format a date in a long format. It is used by the date formatting routines. The initialization routines of the `SysUtils` unit initialize this string with a value conforming to the regional preferences of the user or system regional default.

This constant is deprecated. You should use `DefaultFormatSettings.LongDateFormat` (1643) instead.

`LongDayNames` : `TWeekNameArray` deprecated

`LongDayNames` is an array with the full names of days. It is used by the date formatting routines. The initialization routines of the `SysUtils` unit initialize this string with a value conforming to the regional preferences of the user or system regional default.

The array is indexed by values as returned by the `DayOfWeek` function.

This constant is deprecated. You should use `DefaultFormatSettings.LongDayNames` (1643) instead.

`LongMonthNames` : `TMonthNameArray` deprecated

`LongMonthNames` is an array with the full names of months. It is used by the date formatting routines. The initialization routines of the `SysUtils` unit initialize this string with a value conforming to the regional preferences of the user or system regional default.

This constant is deprecated. You should use `DefaultFormatSettings.LongMonthNames` (1643) instead.

`LongTimeFormat` : string deprecated

`LongTimeFormat` contains a template to format a time in full notation. It is used by the time formatting routines. The initialization routines of the `SysUtils` unit initialize this string with a value conforming to the regional preferences of the user or system regional default.

This constant is deprecated. You should use `DefaultFormatSettings.LongTimeFormat` (1643) instead.

NegCurrFormat : Bytedeprecated

CurrencyFormat is the default format string for negative currencies. It is used by the float formatting routines. The initialization routines of the SysUtils unit initialize this string with a value conforming to the regional preferences of the user or system regional default:

- 0** Left parenthesis, currency symbol, amount, right parenthesis. Ex: (\$1.2)
- 1** Negative sign, currency symbol, amount. Ex: -\$1.2
- 2** Monetary symbol, negative sign, amount. Ex: \$-1.2
- 3** Monetary symbol, amount, negative sign. Ex: \$1.2-
- 4** Left parenthesis, amount, currency symbol, right parenthesis. Ex: (1.2\$)
- 5** Negative sign, amount, currency symbol. Ex: -1.2\$
- 5 6** Amount, negative sign, currency symbol. Ex: 1.2-\$
- 5 7** Amount, currency symbol, negative sign. Ex: 1.2\$-
- 5 8** Negative sign, amount, space, currency symbol (as #5, adding a space before the currency symbol). Ex: -1.2 \$
- 9** Negative sign, currency symbol, space, amount (as #1, adding a space after the currency symbol). Ex: -\$ 1.2
- 10** Amount, space, currency symbol, negative sign (as #7, adding a space before the currency symbol). Ex: 1.2 \$-
- 11** Monetary symbol, space, amount, negative sign (as #3, adding a space after the currency symbol). Ex: \$ 1.2-
- 12** Monetary symbol, space, negative sign, amount (as #2, adding a space after the currency symbol). Ex: \$ -1.2
- 13** Amount, negative sign, space, currency symbol (as #6, adding a space before the currency symbol). Ex: 1.2- \$
- 14** Left parenthesis, currency symbol, space, amount, right parenthesis (as #0, adding a space after the currency symbol). Ex: (\$ 1.2)
- 15** Left parenthesis, amount, space, currency symbol, right parenthesis (as ##4, adding a space before the currency symbol). Ex: (1.2 \$)

OnBeep : TBeepHandler = Nil

OnBeep is called whenever Beep is called. Beep contains no implementation to actually produce a beep, since there is no way to implement beep in a meaningful way for all possible implementations.

OnCreateGUID : TCreateGUIDFunc = Nil

OnCreateGUID can be set to point to a custom routine that creates GUID values. If set, the CreateGUID (1672) function will use it to obtain a GUID value. If it is not set, a default implementation using random values will be used to create the unique value. The function should return a valid GUID in the GUID parameter, and should return zero in case of success.

`OnGetApplicationName : TGetAppNameEvent`

By default, the configuration file routines `GetAppConfigDir` (1726) and `GetAppConfigFile` (1726) use a default application name to construct a directory or filename. This callback can be used to provide an alternative application name.

Since the result of this callback will be used to construct a filename, care should be taken that the returned name does not contain directory separator characters or characters that cannot appear in a filename.

`OnGetTempDir : TGetTempDirEvent`

`OnGetTempDir` can be used to provide custom behaviour for the `GetTempDir` (1732) function. Note that the returned name should have a trailing directory delimiter character.

`OnGetTempFile : TGetTempFileEvent`

`OnGetTempDir` can be used to provide custom behaviour for the `GetTempFileName` (1733) function. Note that the values for `Prefix` and `Dir` should be observed.

`OnGetVendorName : TGetVendorNameEvent`

`OnGetVendorName` must be set in order for `VendorName` (1796) to return a value. It will then be used in `GetAppConfigDir` (1726) and `GetAppConfigFile` (1726) to determine the configuration directory. Set it to a callback that returns the actual vendor name for the application.

`OnShowException : procedure(Msg: ShortString)`

`OnShowException` is the callback that `ShowException` (1751) uses to display a message in a GUI application. For GUI applications, this variable should always be set. Note that no memory may be available when this callback is called, so the callback should already have all resources it needs, when the callback is set.

`ShortDateFormat : stringdeprecated`

`ShortDateFormat` contains a template to format a date in a short format. It is used by the date formatting routines. The initialization routines of the `SysUtils` unit initialize this string with a value conforming to the regional preferences of the user or system regional default.

This constant is deprecated. You should use `DefaultFormatSettings.ShortDateFormat` (1643) instead.

`ShortDayNames : TWeekNameArraydeprecated`

`ShortDayNames` is an array with the abbreviated names of days. It is used by the date formatting routines. The initialization routines of the `SysUtils` unit initialize this string with a value conforming to the regional preferences of the user or system regional default.

The array is indexed by values as returned by the `DayOfWeek` function.

This constant is deprecated. You should use `DefaultFormatSettings.ShortDayNames` (1643) instead.

`ShortMonthNames : TMonthNameArraydeprecated`

`ShortMonthNames` is an array with the abbreviated names of months. It is used by the date formatting routines. The initialization routines of the `SysUtils` unit initialize this string with a value conforming to the regional preferences of the user or system regional default.

This constant is deprecated. You should use `DefaultFormatSettings.ShortMonthNames` (1643) instead.

`ShortTimeFormat` : `stringdeprecated`

`ShortTimeFormat` contains a template to format a time in a short notation. It is used by the time formatting routines. The initialization routines of the `SysUtils` unit initialize this string with a value conforming to the regional preferences of the user or system regional default.

This constant is deprecated. You should use `DefaultFormatSettings.ShortTimeFormat` (1643) instead.

`SysLocale` : `TSysLocale`

`SysLocale` is initialized by the initialization code of the `SysUtils` unit. For an explanation of the fields, see `TSysLocale` (1639)

`ThousandSeparator` : `AnsiChardeprecated`

`ThousandSeparator` is used to separate groups of thousands in floating point numbers or currencies. It is used by the float formatting routines. The initialization routines of the `SysUtils` unit initialize this string with a value conforming to the regional preferences of the user or system regional default.

This constant is deprecated. You should use `DefaultFormatSettings.ThousandSeparator` (1643) instead.

`TimeAMString` : `stringdeprecated`

`TimeAMString` is used to display the AM symbol in the time formatting routines. It is used by the time formatting routines. The initialization routines of the `SysUtils` unit initialize this string with a value conforming to the regional preferences of the user or system regional default.

This constant is deprecated. You should use `DefaultFormatSettings.TimeAMString` (1643) instead.

`TimePMString` : `stringdeprecated`

`TimePMString` is used to display the PM symbol in the time formatting routines. It is used by the time formatting routines. The initialization routines of the `SysUtils` unit initialize this string with a value conforming to the regional preferences of the user or system regional default.

This constant is deprecated. You should use `DefaultFormatSettings.TimePMString` (1643) instead.

`TimeSeparator` : `AnsiChardeprecated`

`TimeSeparator` is used by the time formatting routines to separate the hours from the minutes and the minutes from the seconds. It is used by the time formatting routines. The initialization routines of the `SysUtils` unit initialize this string with a value conforming to the regional preferences of the user or system regional default.

`TrueBoolStrs` : `Array of string`

`TrueBoolStrs` contains the strings that will result in a `True` return value by `StrToBool` (1768).

`TwoDigitYearCenturyWindow` : `Word`

Window to determine what century 2 digit years are in.

76.15 Procedures and functions

76.15.1 AbandonSignalHandler

Synopsis: Abandon the signal handler.

Declaration: `procedure AbandonSignalHandler(RtlSigNum: Integer)`

Visibility: default

Description: `AbandonSignalHandler` tells the system routines that they should not re-install the signal handler for signal `RtlSigNum` under any circumstances. Normally, signal handlers are re-set when they are called. If `AbandonSignalHandler` has been called for a signal that is handled by the system code, the signal will not be re-set again.

76.15.2 Abort

Synopsis: Abort program execution.

Declaration: `procedure Abort`

Visibility: default

Description: `Abort` raises an `EAbort` (1811) exception.

See also: `EAbort` (1811)

76.15.3 AddDisk

Synopsis: Add a disk to the list of known disks (Unix only).

Declaration: `function AddDisk(const path: string) : Byte`

Visibility: default

Description: On Unix-like platforms both the `DiskFree` (1681) and `DiskSize` (1682) functions need a file on the specified drive, since is required for the `statfs` system call.

These filenames are set in `drivestr[0..26]`, and the first 4 have been preset to :

Disk 0 ' . ' default drive - hence current directory is used.

Disk 1 ' /fd0/ . ' floppy drive 1.

Disk 2 ' /fd1/ . ' floppy drive 2.

Disk 3 ' / ' C: equivalent of DOS is the root partition.

Drives 4..26 can be set by your own applications with the `AddDisk` call.

The `AddDisk` call adds `Path` to the names of drive files, and returns the number of the disk that corresponds to this drive. If you add more than 21 drives, the count is wrapped to 4.

Errors: None.

See also: `DiskFree` (1681), `DiskSize` (1682)

76.15.4 AddTerminateProc

Synopsis: Add a procedure to the exit chain.

Declaration: `procedure AddTerminateProc (TermProc: TTerminateProc)`

Visibility: default

Description: `AddTerminateProc` adds `TermProc` to the list of exit procedures. When the program exits, the list of exit procedures is run over, and all procedures are called one by one, in the reverse order that they were added to the exit chain.

Errors: If no memory is available on the heap, an exception may be raised.

See also: `TTerminateProc` ([1640](#)), `CallTerminateProcs` ([1666](#))

76.15.5 AdjustLineBreaks

Synopsis: Convert possible line-endings to the currently valid line ending.

Declaration: `function AdjustLineBreaks(const S: string) : string`
`function AdjustLineBreaks(const S: string; Style: TTextLineBreakStyle)`
`: string`

Visibility: default

Description: `AdjustLineBreaks` will change all occurrences of `#13` and `#10` characters with the correct line-ending characters for the current platform, or the specified platform if the 'Style' parameter was specified.

The platform-specific line-ending characters are:

- `#13#10` on Windows and DOS
- `#13` on Classic Mac OS before OS X
- `#10` on Unix clones / Linux / OS X / macOS.

Errors: None.

See also: `AnsiCompareStr` ([1649](#)), `AnsiCompareText` ([1650](#))

Listing: `./examples/sysutex/ex48.pp`

Program Example48;

{ This program demonstrates the AdjustLineBreaks function }

Uses sysutils;

Const

`S = 'This is a string'#13'with embedded'#10'linefeed and'+`
`#13'CR characters';`

Begin

`Writeln (AdjustLineBreaks(S));`

End.

76.15.6 AnsiCompareFileName

Synopsis: Compare 2 filenames.

Declaration: `function AnsiCompareFileName(const S1: string; const S2: string)
: SizeInt`

Visibility: default

Description: `AnsiCompareFileName` compares 2 filenames `S1` and `S2`, and returns

< 0 if `S1`<`S2`.

= 0 if `S1`=`S2`.

> 0 if `S1`>`S2`.

The function actually checks `FileNameCaseSensitive` and returns the result of `AnsiCompareStr` (1649) or `AnsiCompareText` (1650) depending on whether `FileNameCaseSensitive` is `True` or `False`

Errors: None.

See also: `AnsiCompareStr` (1649), `AnsiCompareText` (1650), `AnsiLowerCaseFileName` (1653)

76.15.7 AnsiCompareStr

Synopsis: Compare 2 ansistrings, case sensitive, using locale settings.

Declaration: `function AnsiCompareStr(const S1: string; const S2: string) : Integer`

Visibility: default

Description: `AnsiCompareStr` compares two strings and returns the following result:

< 0 if `S1`<`S2`.

0 if `S1`=`S2`.

> 0 if `S1`>`S2`.

The comparison takes into account Ansi characters, i.e. it observes the locale. Contrary to `AnsiCompareText` (1650), the comparison is case sensitive.

Remark A widestring manager must be installed in order for this function to work correctly with various character sets.

Errors: None.

See also: `AdjustLineBreaks` (1648), `AnsiCompareText` (1650)

Listing: `./examples/sysutex/ex49.pp`

Program Example49;

```
{ This program demonstrates the AnsiCompareStr function }
{$H+}
```

Uses sysutils;

Procedure TestIt (S1,S2 : String);

Var R : Longint;

```

begin
  R:=AnsiCompareStr(S1,S2);
  Write ('',S1,' is ');
  If R<0 then
    write ('less than ')
  else If R=0 then
    Write ('equal to ')
  else
    Write ('larger than ');
  Writeln ('',S2,'');
end;

Begin
  Testit('One string','One smaller string');
  Testit('One string','one string');
  Testit('One string','One string');
  Testit('One string','One tall string');
End.

```

76.15.8 AnsiCompareText

Synopsis: Compare 2 ansistrings, case insensitive, using locale settings.

Declaration: `function AnsiCompareText(const S1: string; const S2: string) : Integer`

Visibility: default

Description: `AnsiCompareText` compares two strings and returns the following result:

<0 if S1<S2.
 0 if S1=S2.
 >0 if S1>S2.

the comparison takes into account Ansi characters, i.e. it observes the locale. Contrary to `AnsiCompareStr` (1649), the comparison is case insensitive.

Remark A widestring manager must be installed in order for this function to work correctly with various character sets.

Errors: None.

See also: `AdjustLineBreaks` (1648), `AnsiCompareText` (1650)

Listing: `./examples/sysutex/ex50.pp`

Program Example49;

```

{ This program demonstrates the AnsiCompareText function }
{$H+}

```

Uses sysutils;

Procedure TestIt (S1,S2 : String);

Var R : Longint;

```

begin
  R:=AnsiCompareText(S1,S2);
  Write ('',S1,' is ');
  If R<0 then
    write ('less than ')
  else If R=0 then
    Write ('equal to ')
  else
    Write ('larger than ');
  Writeln ('',S2,'');
end;

Begin
  Testit('One string','One smaller string');
  Testit('One string','one string');
  Testit('One string','One string');
  Testit('One string','One tall string');
End.

```

76.15.9 AnsiDequotedStr

Synopsis: Extract string without quotes.

Declaration: `function AnsiDequotedStr(const S: string; AQuote: Char) : string`

Visibility: default

Description: `AnsiDequotedStr` is, similar to `AnsiExtractQuotedStr` ([1651](#)), a method to extract a string that was quoted using `AnsiQuotedStr` ([1654](#)). The string `S` must start and end with the quote character `AQuote` (they will be removed from the result) and all double occurrences of the quote character `AQuote` will be reduced to a single quote character.

Errors: If the string does not start and end with the quote character or has length less than 2, the original string is returned.

See also: `AnsiExtractQuotedStr` ([1651](#)), `AnsiQuotedStr` ([1654](#))

76.15.10 AnsiExtractQuotedStr

Synopsis: Removes the first quoted string from a string.

Declaration: `function AnsiExtractQuotedStr(var Src: PAnsiChar; Quote: AnsiChar) : AnsiString`
`function AnsiExtractQuotedStr(var Src: PWideChar; Quote: WideChar) : WideString`
`function AnsiExtractQuotedStr(var Src: PWideChar; Quote: AnsiChar) : WideString`

Visibility: default

Description: `AnsiExtractQuotedStr` returns the first quoted string in `Src`, and deletes the result from `Src`. The resulting string has with `Quote` characters removed from the beginning and end of the string (if they are present), and double `Quote` characters replaced by a single `Quote` characters. As such, it reverses the action of `AnsiQuotedStr` ([1654](#)).

Errors: None.

See also: `AnsiQuotedStr` ([1654](#))

Listing: `./examples/sysutex/ex51.pp`

```

Program Example51;
{ This program demonstrates the AnsiQuotedStr function }
Uses sysutils;

Var
  S : AnsiString;
  P : PChar;

Begin
  S:='He said "Hello" and walked on';
  P:=Pchar(S);
  S:=AnsiQuotedStr(P,"");
  Writeln (S);
  P:=Pchar(S);
  Writeln(AnsiExtractQuotedStr(P,""));
End.

```

76.15.11 AnsiLastChar

Synopsis: Return a pointer to the last character of a string.

Declaration: `function AnsiLastChar(const S: AnsiString) : PAnsiChar`
`function AnsiLastChar(const S: UnicodeString) : PWideChar`

Visibility: default

Description: This function returns a pointer to the last character of S.

Remark A widestring manager must be installed in order for this function to work correctly with various character sets. If none is installed, this function is the same as `@S[Length[S]]`.

Errors: None.

See also: `AnsiStrLastChar` ([1656](#))

Listing: `./examples/sysutex/ex52.pp`

```

Program Example52;

{ This program demonstrates the AnsiLastChar function }

Uses sysutils;

Var S : AnsiString;
    L : Longint;

Begin
  S:='This is an ansistring.';
  Writeln ('Last character of S is : ',AnsiLastChar(S));
  L:=Longint(AnsiLastChar(S))-Longint(@S[1])+1;
  Writeln ('Length of S is : ',L);
End.

```

76.15.12 AnsiLowerCase

Synopsis: Return a lowercase version of a string.

Declaration: `function AnsiLowerCase(const s: string) : string`

Visibility: default

Description: `AnsiLowerCase` converts the string `S` to lowercase characters and returns the resulting string. It takes into account the operating system language settings when doing this, so special characters are converted correctly as well.

Remark A widestring manager must be installed in order for this function to work correctly with various character sets.

Errors: None.

See also: `AnsiUpperCase` ([1661](#)), `AnsiStrLower` ([1659](#)), `AnsiStrUpper` ([1660](#))

Listing: `./examples/sysutex/ex53.pp`

Program Example53;

{ This program demonstrates the AnsiLowerCase function }

Uses sysutils;

Procedure Testit (S : **String**);

begin
 Writeln (S, ' -> ', **AnsiLowerCase**(S))
end;

Begin
 Testit('AN UPPERCASE STRING');
 Testit('Some mixed STring');
 Testit('a lowercase string');
End.

76.15.13 AnsiLowerCaseFileName

Synopsis: Convert filename to lowercase.

Declaration: `function AnsiLowerCaseFileName(const s: string) : string`

Visibility: default

Description: `AnsiLowerCaseFileName` simply returns the result of

`AnsiLowerCase(S);`

See also: `AnsiLowerCase` ([1653](#)), `AnsiCompareFileName` ([1649](#)), `AnsiUpperCaseFileName` ([1661](#))

76.15.14 AnsiPos

Synopsis: Return Position of one anstring in another.

Declaration: `function AnsiPos(const substr: string; const s: string) : SizeInt`

Visibility: default

Description: `AnsiPos` does the same as the standard `Pos` function.

See also: `AnsiStrPos` ([1659](#)), `AnsiStrScan` ([1660](#)), `AnsiStrRScan` ([1660](#))

76.15.15 AnsiQuotedStr

Synopsis: Return a quoted version of a string.

Declaration: `function AnsiQuotedStr(const S: string; Quote: Char) : string`

Visibility: default

Description: `AnsiQuotedString` quotes the string `S` and returns the result. This means that it puts the `Quote` character at both the beginning and end of the string and replaces any occurrence of `Quote` in `S` with 2 `Quote` characters. The action of `AnsiQuotedString` can be reversed by `AnsiExtractQuotedStr` ([1651](#)).

For an example, see `AnsiExtractQuotedStr` ([1651](#))

Errors: None.

See also: `AnsiExtractQuotedStr` ([1651](#))

76.15.16 AnsiSameStr

Synopsis: Checks whether 2 strings are the same (case sensitive).

Declaration: `function AnsiSameStr(const s1: string; const s2: string) : Boolean`

Visibility: default

Description: `SameText` calls `AnsiCompareStr` ([1649](#)) with `S1` and `S2` as parameters and returns `True` if the result of that call is zero, or `False` otherwise.

Remark A widestring manager must be installed in order for this function to work correctly with various character sets.

Errors: None.

See also: `AnsiCompareStr` ([1649](#)), `SameText` ([1750](#)), `AnsiSameText` ([1654](#))

76.15.17 AnsiSameText

Synopsis: Checks whether 2 strings are the same (case insensitive).

Declaration: `function AnsiSameText(const s1: string; const s2: string) : Boolean`

Visibility: default

Description: `SameText` calls `AnsiCompareText` ([1650](#)) with `S1` and `S2` as parameters and returns `True` if the result of that call is zero, or `False` otherwise.

See also: `AnsiCompareText` ([1650](#)), `SameText` ([1750](#)), `AnsiSameStr` ([1654](#))

76.15.18 AnsiStrComp

Synopsis: Compare two null-terminated strings. Case sensitive.

Declaration: `function AnsiStrComp(S1: PAnsiChar; S2: PAnsiChar) : Integer`

Visibility: default

Description: `AnsiStrComp` compares 2 `PChar` strings, and returns the following result:

<0 if `S1<S2`.

0 if `S1=S2`.

>0 if `S1>S2`.

The comparison of the two strings is case-sensitive.

Remark A widestring manager must be installed in order for this function to work correctly with various character sets.

Errors: None.

See also: `AnsiCompareText` ([1650](#)), `AnsiCompareStr` ([1649](#))

Listing: `./examples/sysutex/ex54.pp`

Program Example54;

{ This program demonstrates the AnsiStrComp function }

Uses sysutils;

Procedure TestIt (S1,S2 : Pchar);

Var R : Longint;

begin

 R:=AnsiStrComp(S1,S2);

Write ('"',S1,'" is ');

If R<0 **then**

write ('less than ')

else If R=0 **then**

Write ('equal to ')

else

Write ('larger than ');

Writeln ('"',S2,'"');

end;

Begin

 Testit('One string','One smaller string');

 Testit('One string','one string');

 Testit('One string','One string');

 Testit('One string','One tall string');

End.

76.15.19 AnsiStrlComp

Synopsis: Compare two null-terminated strings. Case insensitive.

Declaration: `function AnsiStrIComp(S1: PAnsiChar; S2: PAnsiChar) : Integer`

Visibility: default

Description: `AnsiStrIComp` compares 2 `PChar` strings, and returns the following result:

`<0` if `S1<S2`.
`0` if `S1=S2`.
`>0` if `S1>S2`.

The comparison of the two strings is case-insensitive.

Remark A widestring manager must be installed in order for this function to work correctly with various character sets.

Errors: None.

See also: `AnsiCompareText` ([1650](#)), `AnsiCompareStr` ([1649](#))

Listing: `./examples/sysutex/ex55.pp`

Program Example55;

{ This program demonstrates the AnsiStrIComp function }

Uses sysutils;

Procedure TestIt (S1,S2 : Pchar);

Var R : Longint;

begin

 R:=AnsiStrIComp(S1,S2);

Write ('"',S1,'" is ');

If R<0 **then**

write ('less than ')

else If R=0 **then**

Write ('equal to ')

else

Write ('larger than ');

Writeln ('"',S2,'"');

end;

Begin

 TestIt('One string','One smaller string');

 TestIt('One string','one string');

 TestIt('One string','One string');

 TestIt('One string','One tall string');

End.

76.15.20 AnsiStrLastChar

Synopsis: Return a pointer to the last character of a string.

Declaration: `function AnsiStrLastChar(Str: PAnsiChar) : PAnsiChar`

Visibility: default

Description: Return a pointer to the last character of the null-terminated string.

Remark A widestring manager must be installed in order for this function to work correctly with various character sets. If none is installed, this function is the same as `@S[Length[S]]`.

Errors: None.

See also: [AnsiCompareText \(1650\)](#), [AnsiCompareStr \(1649\)](#)

Listing: ./examples/sysutex/ex56.pp

Program Example56;

{ This program demonstrates the AnsiStrLComp function }

Uses sysutils;

Procedure TestIt (S1,S2 : Pchar; L : longint);

```
Var R : Longint;
```

begin

```
R:=AnsiStrLComp(S1,S2,L);
```

Write ('First ',L,' characters of "',S1,'" are ');

If $R < 0$ then

write ('less than ')

else If $R=0$ then

Write ('equal to ')

else

Write ('larger than');

WriteIn ('those of "',S2,'"');

end;

Begin

```
Testit('One string','One smaller string',255);
```

```
Testit('One string','One String',4);
```

```
Testit('One string','1 string',0);
```

```
Testit('One string','One string.',9);
```

End.

76.15.21 AnsiStrLComp

Synopsis: Compare a limited number of characters of 2 strings.

```
Declaration: function AnsiStrLComp(S1: PAnsiChar; S2: PAnsiChar; MaxLen: SizeUInt)
              : Integer
```

Visibility: default

Description: `AnsiStrLComp` functions the same as `AnsiStrComp` (1655), but compares at most `MaxLen` characters. If the first `MaxLen` characters in both strings are the same, then zero is returned.

Note that this function processes embedded null characters, treating them as a normal character.

Errors: None.

See also: [AnsiStrComp \(1655\)](#), [AnsiStrIComp \(1655\)](#), [AnsiStrLComp \(1658\)](#)

76.15.22 AnsiStrLIComp

Synopsis: Compares a given number of characters of a string, case insensitive.

Declaration: `function AnsiStrLIComp(S1: PAnsiChar; S2: PAnsiChar; MaxLen: SizeUInt)
: Integer`

Visibility: default

Description: `AnsiStrLIComp` compares the first `Maxlen` characters of 2 `PChar` strings, `S1` and `S2`, and returns the following result:

<0 if `S1`<`S2`.

0 if `S1`=`S2`.

>0 if `S1`>`S2`.

The comparison of the two strings is case-insensitive.

Remark A widestring manager must be installed in order for this function to work correctly with various character sets.

Errors: None.

See also: `AnsiCompareText` ([1650](#)), `AnsiCompareStr` ([1649](#))

Listing: `./examples/sysutex/ex57.pp`

Program Example57;

{ This program demonstrates the AnsiStrLIComp function }

Uses sysutils;

Procedure TestIt (S1,S2 : Pchar; L : longint);

Var R : Longint;

begin

 R:=AnsiStrLIComp(S1,S2,L);

Write ('First ',L,' characters of "',S1,'" are ');

If R<0 **then**

write ('less than ')

else If R=0 **then**

Write ('equal to ')

else

Write ('larger than ');

Writeln ('those of "',S2,'"');

end;

Begin

 TestIt('One string','One smaller string',255);

 TestIt('ONE STRING','one String',4);

 TestIt('One string','1 STRING',0);

 TestIt('One STRING','one string.',9);

End.

76.15.23 AnsiStrLower

Synopsis: Convert a null-terminated string to all-lowercase characters.

Declaration: `function AnsiStrLower(Str: PAnsiChar) : PAnsiChar`

Visibility: default

Description: `AnsiStrLower` converts the `PChar` `Str` to lowercase characters and returns the resulting `pchar`.

Note that `Str` itself is modified, not a copy, as in the case of `AnsiLowerCase` (1653). It takes into account the operating system language settings when doing this, so special characters are converted correctly as well.

Remark A widestring manager must be installed in order for this function to work correctly with various character sets.

Errors: None.

See also: `AnsiStrUpper` (1660), `AnsiLowerCase` (1653)

Listing: `./examples/sysutex/ex59.pp`

Program Example59;

{ This program demonstrates the AnsiStrLower function }

Uses sysutils,strings;

Procedure Testit (S : Pchar);

begin

Writeln (S, ' -> ',AnsiStrLower(**StrNew**(S)))

end;

Begin

 Testit('AN UPPERCASE STRING');

 Testit('Some mixed STring');

 Testit('a lowercase string');

End.

76.15.24 AnsiStrPos

Synopsis: Return position of one null-terminated substring in another.

Declaration: `function AnsiStrPos(str: PAnsiChar; substr: PAnsiChar) : PAnsiChar`

Visibility: default

Description: `AnsiStrPos` returns a pointer to the first occurrence of `SubStr` in `Str`. If `SubStr` does not occur in `Str` then `Nil` is returned.

Errors: An access violation may occur if either `Str` or `SubStr` point to invalid memory.

See also: `AnsiPos` (1654), `AnsiStrScan` (1660), `AnsiStrRScan` (1660)

76.15.25 AnsiStrRScan

Synopsis: Find last occurrence of a character in a null-terminated string.

Declaration: `function AnsiStrRScan(Str: PAnsiChar; Chr: AnsiChar) : PAnsiChar`

Visibility: default

Description: `AnsiStrPos` returns a pointer to the *last* occurrence of the character `Chr` in `Str`. If `Chr` does not occur in `Str` then `Nil` is returned.

Errors: An access violation may occur if `Str` points to invalid memory.

See also: `AnsiPos` ([1654](#)), `AnsiStrScan` ([1660](#)), `AnsiStrPos` ([1659](#))

76.15.26 AnsiStrScan

Synopsis: Find first occurrence of a character in a null-terminated string.

Declaration: `function AnsiStrScan(Str: PAnsiChar; Chr: AnsiChar) : PAnsiChar`

Visibility: default

Description: `AnsiStrPos` returns a pointer to the *first* occurrence of the character `Chr` in `Str`. If `Chr` does not occur in `Str` then `Nil` is returned.

Errors: An access violation may occur if `Str` points to invalid memory.

See also: `AnsiPos` ([1654](#)), `AnsiStrScan` ([1660](#)), `AnsiStrPos` ([1659](#))

76.15.27 AnsiStrUpper

Synopsis: Convert a null-terminated string to all-uppercase characters.

Declaration: `function AnsiStrUpper(Str: PAnsiChar) : PAnsiChar`

Visibility: default

Description: `AnsiStrUpper` converts the `PCharStr` to uppercase characters and returns the resulting string. Note that `Str` itself is modified, not a copy, as in the case of `AnsiUpperCase` ([1661](#)). It takes into account the operating system language settings when doing this, so special characters are converted correctly as well.

Remark A widestring manager must be installed in order for this function to work correctly with various character sets.

Errors: None.

See also: `AnsiUpperCase` ([1661](#)), `AnsiStrLower` ([1659](#)), `AnsiLowerCase` ([1653](#))

Listing: `./examples/sysutex/ex60.pp`

Program Example60;

{ This program demonstrates the AnsiStrUpper function }

Uses sysutils, strings;

Procedure Testit (S : Pchar);

```
begin
  Writeln (S, ' -> ', AnsiStrUpper(StrNew(S)))
end;
```

```
Begin
  Testit('AN UPPERCASE STRING');
  Testit('Some mixed STring');
  Testit('a lowercase string');
End.
```

76.15.28 AnsiUpperCase

Synopsis: Return an uppercase version of a string, taking into account special characters.

Declaration: `function AnsiUpperCase(const s: string) : string`

Visibility: default

Description: `AnsiUpperCase` converts the string `S` to uppercase characters and returns the resulting string. It takes into account the operating system language settings when doing this, so special characters are converted correctly as well.

Remark A widestring manager must be installed in order for this function to work correctly with various character sets.

Errors: None.

See also: `AnsiStrUpper` ([1660](#)), `AnsiStrLower` ([1659](#)), `AnsiLowerCase` ([1653](#))

Listing: `./examples/sysutex/ex61.pp`

Program Example60;

{ This program demonstrates the AnsiUpperCase function }

Uses sysutils;

Procedure Testit (S : **String**);

```
begin
  Writeln (S, ' -> ', AnsiUpperCase(S))
end;
```

```
Begin
  Testit('AN UPPERCASE STRING');
  Testit('Some mixed STring');
  Testit('a lowercase string');
End.
```

76.15.29 AnsiUpperCaseFileName

Synopsis: Convert filename to uppercase.

Declaration: `function AnsiUpperCaseFileName(const s: string) : string`

Visibility: default

Description: `AnsiUpperCaseFileName` simply returns the result of

```
AnsiUpperCase(S);
```

See also: `AnsiUpperCase` ([1661](#)), `AnsiCompareFileName` ([1649](#)), `AnsiLowerCaseFileName` ([1653](#))

76.15.30 AppendStr

Synopsis: Append one ansistring to another.

Declaration: `procedure AppendStr(var Dest: string; const S: string)`

Visibility: default

Description: `AppendStr` appends `S` to `Dest`.

This function is provided for Delphi compatibility only, since it is completely equivalent to `Dest := Dest + S`.

Errors: None.

See also: `AssignStr` ([1663](#)), `NewStr` ([1745](#)), `DisposeStr` ([1682](#))

Listing: `./examples/sysutex/ex62.pp`

Program Example62;

{ This program demonstrates the AppendStr function }

Uses sysutils;

Var S : AnsiString;

Begin

S := 'This is an ';

AppendStr(S, 'AnsiString');

Writeln ('S = "', S, '"');

End.

76.15.31 ApplicationName

Synopsis: Return a default application name.

Declaration: `function ApplicationName : string`

Visibility: default

Description: `ApplicationName` returns the name of the current application. Standard this is equal to the filename part minus extension of `ParamStr(0)`, but it can be customized by setting the `OnGetApplicationName` ([1645](#)) callback.

Note that the returned value is only the name portion. It does not contain any path or file extension.

Errors: None.

See also: `GetAppConfigDir` ([1726](#)), `OnGetApplicationName` ([1645](#)), `GetAppConfigFile` ([1726](#)), `ConfigExtension` ([1617](#))

76.15.32 ArrayOfConstToStr

Declaration: `function ArrayOfConstToStr(const Args: Array of const;
aSeparator: Char; aQuoteBegin: Char;
aQuoteEnd: Char) : UTF8String`

Visibility: default

76.15.33 ArrayOfConstToStrArray

Declaration: `function ArrayOfConstToStrArray(const Args: Array of const)
: TUTF8StringDynArray`

Visibility: default

76.15.34 AssignStr

Synopsis: Assigns an ansistring to a null-terminated string.

Declaration: `procedure AssignStr(var P: PString; const S: string)`

Visibility: default

Description: `AssignStr` allocates `S` to `P`. The old value of `P` is disposed of.

This function is provided for Delphi compatibility only. `AnsiStrings` are managed on the heap and should be preferred to the mechanism of dynamically allocated strings.

Errors: None.

See also: `NewStr` ([1745](#)), `AppendStr` ([1662](#)), `DisposeStr` ([1682](#))

Listing: `./examples/sysutex/ex63.pp`

Program Example63;

{ This program demonstrates the AssignStr function }
{ \$H+ }

Uses sysutils;

Var P : PString;

Begin

P:=**NewStr**('A first AnsiString');
Writeln ('Before: P = "',P^,'"');
AssignStr(P,'A Second ansistring');
Writeln ('After : P = "',P^,'"');
DisposeStr(P);

End.

76.15.35 BCDToInt

Synopsis: Convert a BCD coded integer to a normal integer.

Declaration: `function BCDToInt(Value: Integer) : Integer`

Visibility: default

Description: `BCDToInt` converts a BCD coded integer to a normal integer.

Errors: None.

See also: `StrToInt` ([1773](#)), `IntToStr` ([1738](#))

Listing: `./examples/sysutex/ex64.pp`

Program `Example64`;

{ This program demonstrates the BCDToInt function }

Uses `sysutils`;

Procedure `Testit` (`L` : `longint`);
begin
 Writeln (L, ' -> ', `BCDToInt`(L));
end;

Begin
 `Testit`(10);
 `Testit`(100);
 `TestIt`(23);
End.

76.15.36 Beep

Synopsis: Sound the system bell.

Declaration: `procedure Beep`

Visibility: `default`

Description: `Beep` sounds the system bell, if one is available. The actual beep is produced by the `OnBeep` ([1644](#)) callback. The `Sysutils` unit itself contains no implementation of this call.

76.15.37 BoolToStr

Synopsis: Convert a boolean value to a string.

Declaration: `function BoolToStr`(`B`: `Boolean`; `UseBoolStrs`: `Boolean`) : `string`
 `function BoolToStr`(`B`: `Boolean`; `const TrueS`: `string`;
 `const FalseS`: `string`) : `string`

Visibility: `default`

Description: `BoolToStr` converts the boolean `B` to a string representation. Which string this is depends on the way the function is called: You can provide the strings to use for the `True` or `False` in the parameters `TrueS` and `FalseS`, or you can tell the function to use the first string in the `TrueBoolStrs` ([1646](#)) and `FalseBoolStrs` ([1643](#)) arrays by setting the `UseBoolStrs` parameter to `True`

The default (Delphi compatible) behaviour is to use `-1` for `True` and `0` for `False`.

Errors: None.

See also: `StrToBool` ([1768](#)), `TrueBoolStrs` ([1646](#)), `FalseBoolStrs` ([1643](#))

76.15.38 ByteLength

Synopsis: Length (in bytes) of a unicodestring.

Declaration: `function ByteLength(const S: UnicodeString) : Integer`

Visibility: default

Description: `ByteLength` returns the length of a unicodestring in bytes. This equals the character length of the string (`Length (1609)`) multiplied by the number of bytes per character (2).

See also: `Length (1609)`

76.15.39 BytesOf

Synopsis: Return the bytes of a string in a `TBytes` array.

Declaration: `function BytesOf(const Val: RawByteString) : TBytes`
`function BytesOf(const Val: AnsiChar) : TBytes`
`function BytesOf(const Val: UnicodeString) : TBytes; Overload`
`function BytesOf(const Val: WideChar) : TBytes; Overload`

Visibility: default

Description: `BytesOf` returns a copy of the string's content as an array of bytes. For an empty string, zero bytes are returned (i.e. `length (BytesOf (S) ) = 0`).

See also: `TBytes (1628)`

76.15.40 ByteToCharIndex

Synopsis: Convert a character index in Bytes to an Index in characters.

Declaration: `function ByteToCharIndex(const S: string; Index: SizeInt) : SizeInt`

Visibility: default

Description: `ByteToCharIndex` returns the index (in characters) of the `Index`-th byte in `S`.

Errors: This function does not take into account MBCS yet.

See also: `CharToByteLen (1667)`, `ByteToCharLen (1665)`

76.15.41 ByteToCharLen

Synopsis: Convert a length in bytes to a length in characters.

Declaration: `function ByteToCharLen(const S: string; MaxLen: SizeInt) : SizeInt`

Visibility: default

Description: `ByteToCharLen` returns the number of bytes in `S`, but limits the result to `MaxLen`

Errors: This function does not take into account MBCS yet.

See also: `CharToByteLen (1667)`, `ByteToCharIndex (1665)`

76.15.42 ByteType

Synopsis: Return the type of byte in an ansistring for a multi-byte character set.

Declaration: `function ByteType(const S: string; Index: SizeUInt) : TMbcsByteType`

Visibility: default

Description: `ByteType` returns the type of byte in the ansistring `S` at (1-based) position `Index`.

Errors: No checking on the index is performed.

See also: `TMbcsByteType` ([1635](#)), `StrByteType` ([1754](#))

76.15.43 CallTerminateProcs

Synopsis: Call the exit chain procedures.

Declaration: `function CallTerminateProcs : Boolean`

Visibility: default

Description: `CallTerminateProcs` is run on program exit. It executes all terminate procedures that were added to the exit chain with `AddTerminateProc` ([1648](#)), and does this in reverse order.

Errors: If one of the exit procedure raises an exception, it is *not* caught, and the remaining exit procedures will not be executed.

See also: `TTerminateProc` ([1640](#)), `AddTerminateProc` ([1648](#))

76.15.44 ChangeFileExt

Synopsis: Change the extension of a filename.

Declaration: `function ChangeFileExt(const FileName: UnicodeString;
 const Extension: UnicodeString) : UnicodeString
function ChangeFileExt(const FileName: RawByteString;
 const Extension: RawByteString) : RawByteString`

Visibility: default

Description: `ChangeFileExt` changes the file extension in `FileName` to `Extension`. The extension `Extension` includes the starting . (dot). The previous extension of `FileName` are all characters after the last ., the . character included.

If `FileName` doesn't have an extension, `Extension` is just appended.

If `Extension` is empty, the effect of this function is to remove any existing extension.

Errors: None.

See also: `ExtractFileExt` ([1691](#)), `ExtractFileName` ([1691](#)), `ExtractFilePath` ([1692](#)), `ExpandFileName` ([1688](#))

76.15.45 ChangeFilePath

Declaration: `function ChangeFilePath(const aFileName: UnicodeString;
 const aPath: UnicodeString) : UnicodeString
function ChangeFilePath(const aFileName: RawByteString;
 const aPath: RawByteString) : RawByteString`

Visibility: default

76.15.46 CharInSet

Synopsis: Check whether a char is in a set of characters.

Declaration: `function CharInSet (Ch: AnsiChar; const CSet: TSysCharSet) : Boolean`
`function CharInSet (Ch: WideChar; const CSet: TSysCharSet) : Boolean`

Visibility: default

Description: `CharInSet` returns `True` if `Ch` matches one of the characters in `CSet`, it returns `False` otherwise. It is equivalent to

`Ch in CSet`

Later versions of this function may take `WideChar` into account.

76.15.47 CharToByteLen

Synopsis: Convert a length in characters to a length in bytes.

Declaration: `function CharToByteLen (const S: string; MaxLen: SizeInt) : SizeInt`

Visibility: default

Description: `CharToByteLen` returns the number of bytes in `S`, but limits the result to `MaxLen`

Errors: This function does not take into account MBCS yet.

See also: `ByteToCharLen` ([1665](#)), `ByteToCharIndex` ([1665](#))

76.15.48 CheckOSError

Synopsis: Check for OS error and raise exception.

Declaration: `procedure CheckOSError (LastError: Integer)`

Visibility: default

Description: `CheckOSError` checks if `LastError` is non-zero and calls `RaiseLastOSError` ([1747](#)) if this is the case.

See also: `RaiseLastOSError` ([1747](#))

76.15.49 CodePageNameToCodePage

Synopsis: Return a numeric identifier for the codepage.

Declaration: `function CodePageNameToCodePage (const cpname: AnsiString)`
`: TSystemCodePage`

Visibility: default

Description: `CodePageNameToCodePage` returns the code page number for the specified codepage `cpname`.

Errors: If the code page is not found in the list of code pages, `$FFFF` is returned.

See also: `CodePageToCodePageName` ([1668](#))

76.15.50 CodePageToCodePageName

Synopsis: Convert a numeric codepage identifier to a codepage name.

Declaration: `function CodePageToCodePageName(cp: TSystemCodePage) : AnsiString`

Visibility: default

Description: `CodePageToCodePageName` returns the name of the codepage `cp`.

Errors: If no matching codepage is found in the list of codepages, an empty string is returned.

See also: `CodePageNameToCodePage` ([1667](#))

76.15.51 CompareMem

Synopsis: Compare two memory areas.

Declaration: `function CompareMem(P1: Pointer; P2: Pointer; &Length: PtrUInt)
: Boolean`

Visibility: default

Description: `CompareMem` compares, byte by byte, 2 memory areas pointed to by `P1` and `P2`, for a length of `L` bytes.

The function returns `True` if all `L` bytes are the same, and `False` otherwise.

76.15.52 CompareMemRange

Synopsis: Compare 2 memory locations.

Declaration: `function CompareMemRange(P1: Pointer; P2: Pointer; &Length: PtrUInt)
: Integer`

Visibility: default

Description: `CompareMemRange` compares the 2 memory locations pointed to by `P1` and `P2` byte per byte. It stops comparing after `Length` bytes have been compared, or when it has encountered 2 different bytes. The result is then

>0 if a byte in range `P1` was found that is bigger than the corresponding byte in range `P2`.

0 if all bytes in range `P1` are the same as the corresponding bytes in range `P2`.

<0 if a byte in range `P1` was found that is less than the corresponding byte in range `P2`.

Errors: None.

See also: `SameText` ([1750](#))

76.15.53 CompareStr

Synopsis: Compare 2 ansistrings case-sensitively, ignoring special characters.

Declaration: `function CompareStr(const S1: string; const S2: string) : Integer
; Overload
function CompareStr(const S1: string; const S2: string;
LocaleOptions: TLocaleOptions) : Integer; Overload`

Visibility: default

Description: CompareStr compares two strings, S1 and S2, and returns the following result:

 ≤ 0 if $s_1 \leq s_2$.

0 if $s_1=s_2$.

>0 if $s_1 > s_2$.

The comparison of the two strings is case-sensitive. The function does not take internationalization settings into account, it simply compares ASCII values.

Errors: None.

See also: [AnsiCompareText \(1650\)](#), [AnsiCompareStr \(1649\)](#), [CompareText \(1669\)](#)

Listing: ./examples/sysutex/ex65.pp

Program Example65;

```
{ This program demonstrates the CompareStr function }
{$H+}
```

Uses sysutils;

```
Procedure TestIt (S1,S2 : String);
```

```
Var R : Longint;
```

begin

```
R:=CompareStr(S1,S2);
```

Write ("",S1,"" is ');

If $R < 0$ then

write ('less than ')

else If $R=0$ then

Write ('equal to ')

else

Write ('larger than ');

```
WriteIn ("",S2,"");
```

end;

Begin

```
Testit('One string','One smaller string');
```

```
Testit('One string','one string');
```

```
Testit('One string','One string');
```

```
Testit('One string','One tall string');
```

End.

76.15.54 CompareText

Synopsis: Compare 2 ansistrings case insensitive.

```
Declaration: function CompareText(const S1: string; const S2: string) : Integer
                                ; Overload
```

```
function CompareText(const S1: string; const S2: string;
                    LocaleOptions: TLocaleOptions) : Integer; Overload
```

Visibility: default

Description: CompareText compares two strings, S1 and S2, and returns the following result:

<0 if S1<S2.

0 if S1=S2.

>0 if S1>S2.

The comparison of the two strings is case-insensitive. The function does not take internationalization settings into account, it simply compares ASCII values.

Errors: None.

See also: AnsiCompareText ([1650](#)), AnsiCompareStr ([1649](#)), CompareStr ([1668](#))

Listing: ./examples/sysutex/ex66.pp

Program Example66;

```
{ This program demonstrates the CompareText function }
{$H+}
```

Uses sysutils;

Procedure TestIt (S1,S2 : String);

Var R : Longint;

begin

 R:=CompareText(S1,S2);

Write ('"',S1,'" is ');

If R<0 **then**

write ('less than ')

else If R=0 **then**

Write ('equal to ')

else

Write ('larger than ');

Writeln ('"',S2,'"');

end;

Begin

 TestIt('One string','One smaller string');

 TestIt('One string','one string');

 TestIt('One string','One string');

 TestIt('One string','One tall string');

End.

76.15.55 ComposeDateTime

Synopsis: Add a date and time.

Declaration: function ComposeDateTime (Date: TDateTime; Time: TDateTime) : TDateTime

Visibility: default

Description: ComposeDateTime correctly adds Date and Time, also for dates before 1899-12-31. For dates after this date, it is just the mathematical addition.

Errors: None.

See also: #rtl.dateutils.EncodeDateTime ([629](#))

76.15.56 ConcatPaths

Synopsis: Concatenate an array of paths to form a single path.

Declaration: `function ConcatPaths(const Paths: Array of UnicodeString) : UnicodeString`
`function ConcatPaths(const Paths: Array of RawByteString) : RawByteString`

Visibility: default

Description: `ConcatPaths` will concatenate the different path components in `Paths` to a single path. It will insert directory separators between the various components of the path as needed. No directory separators will be added to the beginning or the end of the path, and none will be taken away.

See also: `IncludeTrailingPathDelimiter` (1736), `IncludeLeadingPathDelimiter` (1736), `ExcludeTrailingPathDelimiter` (1687), `IncludeTrailingPathDelimiter` (1736)

Listing: `./examples/sysutex/ex96.pp`

program ex96;

{ This program demonstrates the Concatpaths function }

uses sysutils;

begin

// will write /this/path/more/levels/

Writeln(ConcatPaths(['/this/', 'path', 'more/levels/']));

// will write this/path/more/levels/

Writeln(ConcatPaths(['this/', 'path', 'more/levels/']));

// will write this/path/more/levels

Writeln(ConcatPaths(['this/', 'path', 'more/levels/']));

end.

76.15.57 CreateDir

Synopsis: Create a new directory.

Declaration: `function CreateDir(const NewDir: RawByteString) : Boolean`
`function CreateDir(const NewDir: UnicodeString) : Boolean`

Visibility: default

Description: `CreateDir` creates a new directory with name `NewDir`. If the directory doesn't contain an absolute path, then the directory is created below the current working directory.

The function returns `True` if the directory was successfully created, `False` otherwise.

Errors: In case of an error, the function returns `False`.

See also: `RemoveDir` (1747)

Listing: `./examples/sysutex/ex26.pp`

Program Example26;

{ This program demonstrates the CreateDir and RemoveDir functions }

{ Run this program twice in the same directory }

Uses sysutils;

Begin

```

If Not DirectoryExists('NewDir') then
  If Not CreateDir ('NewDir') Then
    Writeln ('Failed to create directory !')
  else
    Writeln ('Created "NewDir" directory')
Else
  If Not RemoveDir ('NewDir') Then
    Writeln ('Failed to remove directory !')
  else
    Writeln ('Removed "NewDir" directory');

```

End.

76.15.58 CreateGUID

Synopsis: Create a new GUID.

Declaration: `function CreateGUID(out GUID: TGuid) : Integer`

Visibility: default

Description: `CreateGUID` can be called to create a new GUID (Globally Unique Identifier) value. The function returns the new GUID value in `GUID` and returns zero in case the GUID was created successfully. If no GUID was created, a nonzero error code is returned.

The default mechanism for creating a new GUID is system dependent. If operating system support is available, it is used. If none is available, a default implementation using random numbers is used.

The `OnCreateGUID` callback can be set to hook a custom mechanism behind the `CreateGUID` function. This can be used to let the GUID be created by an external GUID creation library.

Errors: On error, a nonzero return value is returned.

See also: `GUIDCase` ([1734](#)), `IsEqualGUID` ([1739](#)), `StringToGUID` ([1759](#)), `TryStringToGUID` ([1786](#)), `GUIDToString` ([1734](#))

76.15.59 CurrentYear

Synopsis: Return the current year.

Declaration: `function CurrentYear : Word`

Visibility: default

Description: `CurrentYear` returns the current year as a 4-digit number.

Errors: None.

See also: `Date` ([1673](#)), `Time` ([1781](#)), `Now` ([1745](#))

76.15.60 CurrToStr

Synopsis: Convert a currency value to a string.

Declaration: `function CurrToStr(Value: Currency) : string`
`function CurrToStr(Value: Currency;`
`const FormatSettings: TFormatSettings) : string`

Visibility: default

Description: `CurrToStr` will convert a currency value to a string with a maximum of 15 digits, and precision 2. Calling `CurrToStr` is equivalent to calling `FloatToStrF` (1710):

```
FloatToStrF(Value, ffNumber, 15, 2);
```

Note that on unix systems, the localization support must be enabled explicitly, see Localization (1610).

Errors: None.

See also: `FloatToStrF` (1710), `StrToCurr` (1768)

76.15.61 CurrToStrF

Synopsis: Format a currency to a string.

Declaration: `function CurrToStrF(Value: Currency; Format: TFloatFormat;`
`Digits: Integer) : string`
`function CurrToStrF(Value: Currency; Format: TFloatFormat;`
`Digits: Integer;`
`const FormatSettings: TFormatSettings) : string`

Visibility: default

Description: `CurrToStrF` formats the currency `Value` according to the value in `Format`, using the number of digits specified in `Digits`, and a precision of 19. This function simply calls `FloatToStrF` (1710).

Note that on unix systems, the localization support must be enabled explicitly, see Localization (1610).

See also: `FloatToStrF` (1710), Localization (1610)

76.15.62 Date

Synopsis: Return the current date.

Declaration: `function Date : TDateTime`

Visibility: default

Description: `Date` returns the current date in `TDateTime` format.

Errors: None.

See also: `Time` (1781), `Now` (1745)

Listing: `./examples/sysutex/ex1.pp`

Program Example1;

{ This program demonstrates the Date function }

uses sysutils;

Var YY,MM,DD : Word;

Begin

WriteLn ('Date : ',**Date**);

DecodeDate (**Date**,YY,MM,DD);

WriteLn (**format** ('Date is (DD/MM/YY): %d/%d/%d ',[dd,mm,yy]));

End.

76.15.63 DateTimeToFileDate

Synopsis: Convert a `TDateTime` value to a file age (integer).

Declaration: `function DateTimeToFileDate(DateTime: TDateTime) : Int64`

Visibility: default

Description: `DateTimeToFileDate` function converts a date/time indication in `TDateTime` format to a file-date function, such as returned for instance by the `FileAge` (1693) function.

Errors: None.

See also: `Time` (1781), `Date` (1673), `FileDateToDateTime` (1696), `DateTimeToSystemTime` (1676), `Date-TimeToTimeStamp` (1677)

Listing: `./examples/sysutex/ex2.pp`

Program Example2;

{ This program demonstrates the DateTimeToFileDate function }

Uses sysutils;

Begin

WriteLn ('FileTime of now would be: ',**DateTimeToFileDate** (**Now**));

End.

76.15.64 DateTimeToStr

Synopsis: Converts a `TDateTime` value to a string using a predefined format.

Declaration: `function DateTimeToStr(DateTime: TDateTime; ForceTimeIfZero: Boolean) : string`
`function DateTimeToStr(DateTime: TDateTime;`
`const FormatSettings: TFormatSettings;`
`ForceTimeIfZero: Boolean) : string`

Visibility: default

Description: `DateTimeToStr` returns a string representation of `DateTime` using the formatting specified in `ShortDateFormat` and `LongTimeFormat`. It corresponds to a call to `FormatDateTime('c', DateTime)` (see `formatchars` (1615)).

Note that if the time part is 0 (i.e. midnight), no time is appended.

Note that on unix systems, the localization support must be enabled explicitly, see `Localization` (1610).

Errors: None.

See also: `FormatDateTime` (1723), `DefaultFormatSettings` (1643), `Localization` (1610)

Listing: `./examples/sysutex/ex3.pp`

Program Example3;

{ This program demonstrates the DateTimeToStr function }

Uses sysutils;

Begin

Writeln ('Today is : ', `DateTimeToStr(Now)`);

Writeln ('Today is : ', `FormatDateTime('c', Now)`);

End.

76.15.65 DateTimeToString

Synopsis: Converts a `TDateTime` value to a string with a given format.

Declaration:

```
procedure DateTimeToString(out Result: string; const FormatStr: string;
                           const DateTime: TDateTime;
                           Options: TFormatDateTimeOptions)
procedure DateTimeToString(out Result: string; const FormatStr: string;
                           const DateTime: TDateTime;
                           const FormatSettings: TFormatSettings;
                           Options: TFormatDateTimeOptions)
```

Visibility: default

Description: `DateTimeToString` returns in `Result` a string representation of `DateTime` using the formatting specified in `FormatStr`.

for a list of characters that can be used in the `FormatStr` formatting string, see `formatchars` (1615).

Note that for 'c', if the time part is 0 (i.e. midnight), no time is appended.

Note that on unix systems, the localization support must be enabled explicitly, see `Localization` (1610).

Errors: In case a wrong formatting character is found, an `EConvertError` (1812) is raised.

See also: `FormatDateTime` (1723), `formatchars` (1615), `EConvertError` (1812), `Localization` (1610)

Listing: `./examples/sysutex/ex4.pp`

Program Example4;

{ This program demonstrates the DateTimeToString function }

Uses sysutils;

Procedure today (Fmt : **string**);

Var S : AnsiString;

```
begin
  DateTimeToString (S,Fmt,Date);
  Writeln (S);
end;
```

Procedure Now (Fmt : **string**);

Var S : AnsiString;

```
begin
  DateTimeToString (S,Fmt,Time);
  Writeln (S);
end;
```

```
Begin
  Today ("Today is "dddd dd mmmm y');
  Today ("Today is "d mmm yy');
  Today ("Today is "d/mmm/yy');
  Now ("The time is "am/pmh:n:s');
  Now ("The time is "hh:nn:ssam/pm');
  Now ("The time is "tt');
End.
```

76.15.66 DateTimeToSystemTime

Synopsis: Converts a TDateTime value to a systemtime structure.

Declaration: `procedure DateTimeToSystemTime (DateTime: TDateTime;
out SystemTime: TSystemTime)`

Visibility: default

Description: DateTimeToSystemTime converts a date/time pair in DateTime, with TDateTime format to a system time SystemTime.

Errors: None.

See also: DateTimeToFileDate ([1674](#)), SystemTimeToDateTime ([1780](#)), DateTimeToTimeStamp ([1677](#))

Listing: ./examples/sysutex/ex5.pp

Program Example5;

{ This program demonstrates the DateTimeToSystemTime function }

Uses sysutils;

Var ST : TSystemTime;

```
Begin
  DateTimeToSystemTime(Now,ST);
```

```

With St do
  begin
    Writeln ('Today is ',year,'/',month,'/',Day);
    Writeln ('The time is ',Hour,':',minute,':',Second,':',MilliSecond);
  end;
End.

```

76.15.67 DateTimeToTimeStamp

Synopsis: Converts a `TDateTime` value to a `TimeStamp` structure.

Declaration: `function DateTimeToTimeStamp (Date: TDateTime) : TTimeStamp`

Visibility: default

Description: `DateTimeToSystemTime` converts a date/time pair in `Date`, with `TDateTime` format to a `TimeStamp` format.

Errors: None.

See also: `DateToFileDate` ([1674](#)), `SystemTimeToDateTime` ([1780](#)), `DateTimeToSystemTime` ([1676](#))

Listing: `./examples/sysutex/ex6.pp`

Program Example6;

{ This program demonstrates the DateTimeToTimeStamp function }

Uses sysutils;

Var TS : TTimeStamp;

Begin

TS:=DateTimeToTimeStamp (**Now**);

With TS **do**

begin

Writeln ('Now is ',time,' millisecond past midnight');

Writeln ('Today is ',Date,' days past 1/1/0001');

end;

End.

76.15.68 DateToStr

Synopsis: Converts a `TDateTime` value to a date string with a predefined format.

Declaration: `function DateToStr (Date: TDateTime) : string`
`function DateToStr (Date: TDateTime;`
`const FormatSettings: TFormatSettings) : string`

Visibility: default

Description: `DateToStr` converts `Date` to a string representation. It uses `ShortDateFormat` as it's formatting string. It is hence completely equivalent to a `FormatDateTime ('dddd', Date)`.

Note that on unix systems, the localization support must be enabled explicitly, see `Localization` ([1610](#)).

Errors: None.

See also: [TimeToStr \(1783\)](#), [DateTimeToStr \(1674\)](#), [FormatDateTime \(1723\)](#), [StrToDate \(1769\)](#), [Localization \(1610\)](#)

Listing: ./examples/sysutex/ex7.pp

Program Example7;

{ This program demonstrates the DateToStr function }

Uses sysutils;

Begin

Writeln(**Format** ('Today is: %s',[**DateToStr**(**Date**)]));

End.

76.15.69 DayOfWeek

Synopsis: Returns the day of the week.

Declaration: `function DayOfWeek (DateTime: TDateTime) : Integer`

Visibility: default

Description: `DayOfWeek` returns the day of the week from `DateTime`. Sunday is counted as day 1, Saturday is counted as day 7. The result of `DayOfWeek` can serve as an index to the `LongDayNames` constant array, to retrieve the name of the day.

Errors: None.

See also: [Date \(1673\)](#), [DateToStr \(1677\)](#)

Listing: ./examples/sysutex/ex8.pp

Program Example8;

{ This program demonstrates the DayOfWeek function }

Uses sysutils;

Begin

Writeln ('Today's day is ',`LongDayNames`[**DayOfWeek**(**Date**)]);

End.

76.15.70 DecodeDate

Synopsis: Decode a `TDateTime` to a year,month,day triplet.

Declaration: `procedure DecodeDate (Date: TDateTime; out Year: Word; out Month: Word; out Day: Word)`

Visibility: default

Description: `DecodeDate` decodes the Year, Month and Day stored in `Date`, and returns them in the Year, Month and Day variables.

Errors: None.

See also: [EncodeDate \(1683\)](#), [DecodeTime \(1679\)](#)

Listing: ./examples/sysutex/ex9.pp

Program Example9;

{ This program demonstrates the DecodeDate function }

Uses sysutils;

Var YY,MM,DD : Word;

Begin

DecodeDate(Date,YY,MM,DD);

Writeln (**Format** ('Today is %d/%d/%d',[dd,mm,yy]));

End.

76.15.71 DecodeDateFully

Synopsis: Decode a date with additional date of the week.

Declaration: `function DecodeDateFully(const DateTime: TDateTime; out Year: Word;
 out Month: Word; out Day: Word; out DOW: Word)
 : Boolean`

Visibility: default

Description: `DecodeDateFully`, like `DecodeDate` ([1678](#)), decodes `DateTime` in its parts and returns these in `Year`, `Month`, `Day` but in addition returns the day of the week in `DOW`.

Errors: None.

See also: [EncodeDate \(1683\)](#), [TryEncodeDate \(1785\)](#), [DecodeDate \(1678\)](#)

76.15.72 DecodeTime

Synopsis: Decode a `TDateTime` to a hour,minute,second,millisecond quartet.

Declaration: `procedure DecodeTime(Time: TDateTime; out Hour: Word; out Minute: Word;
 out Second: Word; out MilliSecond: Word)`

Visibility: default

Description: `DecodeTime` decodes the hours, minutes, second and milliseconds stored in `Time`, and returns them in the `Hour`, `Minute` and `Second` and `MilliSecond` variables.

Errors: None.

See also: [EncodeTime \(1684\)](#), [DecodeDate \(1678\)](#)

Listing: ./examples/sysutex/ex10.pp

Program Example10;

{ This program demonstrates the DecodeTime function }

Uses sysutils;

Var HH,MM,SS,MS: Word;

Begin

DecodeTime(Time,HH,MM,SS,MS);

Writeln (format('The time is %d:%d:%d.%d',[hh,mm,ss,ms]));

End.

76.15.73 DeleteFile

Synopsis: Delete a file from the file system.

Declaration: `function DeleteFile(const FileName: UnicodeString) : Boolean`
 `function DeleteFile(const FileName: RawByteString) : Boolean`

Visibility: default

Description: DeleteFile deletes file FileName from disk. The function returns True if the file was successfully removed, False otherwise.

Errors: On error, False is returned.

See also: FileCreate ([1694](#)), FileExists ([1697](#))

Listing: ./examples/sysutex/ex31.pp

Program Example31;

{ This program demonstrates the DeleteFile function }

Uses sysutils;

Var

 Line : **String**;

 F,I : Longint;

Begin

 F:=FileCreate('test.txt');

 Line:='Some string line.'#10;

For I:=1 **to** 10 **do**

 FileWrite (F,Line[1],Length(Line));

FileClose(F);

DeleteFile('test.txt');

End.

76.15.74 DirectoryExists

Synopsis: Check whether a directory exists in the file system.

Declaration: `function DirectoryExists(const Directory: UnicodeString;
FollowLink: Boolean) : Boolean
function DirectoryExists(const Directory: RawByteString;
FollowLink: Boolean) : Boolean`

Visibility: default

Description: `DirectoryExists` checks whether `Directory` exists in the file system and is actually a directory. If this is the case, the function returns `True`, otherwise `False` is returned.

See also: `FileExists` ([1697](#))

76.15.75 DiskFree

Synopsis: Return the amount of free disk space.

Declaration: `function DiskFree(drive: Byte) : Int64`

Visibility: default

Description: `DiskFree` returns the free space (in bytes) on disk `Drive`. `Drive` is the number of the disk drive:

- 0** for the current drive.
- 1** for the first floppy drive.
- 2** for the second floppy drive.
- 3** for the first hard-disk partition.
- 4-26** for all other drives and partitions.

Remark Under Linux, and Unix in general, the concept of disk is different than the dos one, since the file system is seen as one big directory tree. For this reason, the `DiskFree` and `DiskSize` ([1682](#)) functions must be mimicked using filenames that reside on the partitions. For more information, see `AddDisk` ([1647](#)).

Errors: On error, `-1` is returned.

See also: `DiskSize` ([1682](#)), `AddDisk` ([1647](#))

Listing: `./examples/sysutex/ex27.pp`

Program Example27;

{ This program demonstrates the DiskFree function }

Uses sysutils;

Begin

```
Write ('Size of current disk : ',DiskSize(0));
Writeln (' (= ',DiskSize(0) div 1024,'k');
Write ('Free space of current disk : ',Diskfree(0));
Writeln (' (= ',Diskfree(0) div 1024,'k');
```

End.

76.15.76 DiskSize

Synopsis: Return the total amount of disk space.

Declaration: `function DiskSize(drive: Byte) : Int64`

Visibility: default

Description: `DiskSize` returns the size (in bytes) of disk `Drive`. `Drive` is the number of the disk drive:

- 0** for the current drive.
- 1** for the first floppy drive.
- 2** for the second floppy drive.
- 3** for the first hard-disk partition.
- 4-26** for all other drives and partitions.

Remark Under Linux, and Unix in general, the concept of disk is different than the dos one, since the file system is seen as one big directory tree. For this reason, the `DiskFree` (1681) and `DiskSize` functions must be mimicked using filenames that reside on the partitions. For more information, see `AddDisk` (1647)

For an example, see `DiskFree` (1681).

Errors: On error, -1 is returned.

See also: `DiskFree` (1681), `AddDisk` (1647)

76.15.77 DisposeStr

Synopsis: Dispose an ansistring from the heap.

Declaration: `procedure DisposeStr(S: PString); Overload`
`procedure DisposeStr(S: PShortString); Overload`

Visibility: default

Description: `DisposeStr` removes the dynamically allocated string `S` from the heap, and releases the occupied memory.

This function is provided for Delphi compatibility only. `AnsiStrings` are managed on the heap and should be preferred to the mechanism of dynamically allocated strings.

For an example, see `DisposeStr` (1682).

Errors: None.

See also: `NewStr` (1745), `AppendStr` (1662), `AssignStr` (1663)

76.15.78 DoDirSeparators

Synopsis: Convert known directory separators to the current directory separator.

Declaration: `procedure DoDirSeparators(var FileName: UnicodeString)`
`procedure DoDirSeparators(var FileName: RawByteString)`

Visibility: default

Description: This function replaces all known directory separators in `FileName` to the directory separator character for the current system. The list of known separators is specified in the `AllowDirectorySeparators` (1342) constant.

Errors: None.

See also: ExtractFileName ([1691](#)), ExtractFilePath ([1692](#))

Listing: ./examples/sysutex/ex32.pp

Program Example32;

{ This program demonstrates the DoDirSeparators function }
{\$H+}

Uses sysutils;

Procedure Testit (F : **String**);

begin

Writeln ('Before : ',F);

 DoDirSeparators (F);

Writeln ('After : ',F);

end;

Begin

 Testit (GetCurrentDir);

 Testit ('c:\pp\bin\win32');

 Testit ('/usr/lib/fpc');

 Testit ('usr/lib/fpc');

End.

76.15.79 EncodeDate

Synopsis: Encode a Year,Month,Day to a TDateTime value.

Declaration: function EncodeDate (Year: Word; Month: Word; Day: Word) : TDateTime

Visibility: default

Description: EncodeDate encodes the Year, Month and Day variables to a date in TDateTime format. It does the opposite of the DecodeDate ([1678](#)) procedure.

The parameters must lie withing valid ranges (boundaries included):

Year must be between 1 and 9999.

Month must be within the range 1-12.

Day must be between 1 and 31.

Errors: In case one of the parameters is out of it's valid range, an EConvertError ([1812](#)) exception is raised.

See also: EncodeTime ([1684](#)), DecodeDate ([1678](#))

Listing: ./examples/sysutex/ex11.pp

Program Example11;

{ This program demonstrates the EncodeDate function }

Uses sysutils;

Var YY,MM,DD : Word;

```

Begin
  DecodeDate (Date,YY,MM,DD);
  WriteLn ('Today is : ',FormatDateTime ('dd mmmm yyyy',EnCodeDate(YY,Mm,Dd)));
End.

```

76.15.80 EncodeTime

Synopsis: Encode a Hour,Min,Sec,millisecond to a TDateTime value.

Declaration: `function EncodeTime (Hour: Word; Minute: Word; Second: Word; MilliSecond: Word) : TDateTime`

Visibility: default

Description: EncodeTime encodes the Hour, Minute, Second, MilliSecond variables to a TDateTime format result. It does the opposite of the DecodeTime (1679) procedure.

The parameters must have a valid range (boundaries included):

Hour must be between 0 and 23.

Minute,second must both be between 0 and 59.

Millisecond must be between 0 and 999.

Errors: In case one of the parameters is out of it's valid range, an EConvertError (1812) exception is raised.

See also: EncodeDate (1683), DecodeTime (1679)

Listing: ./examples/sysutex/ex12.pp

Program Example12;

{ This program demonstrates the EncodeTime function }

Uses sysutils;

Var Hh,MM,SS,MS : Word;

```

Begin
  DeCodeTime (Time,Hh,MM,SS,MS);
  WriteLn ('Present Time is : ',FormatDateTime('hh:mm:ss',EnCodeTime (Hh,MM,SS,MS)));
End.

```

76.15.81 ExceptAddr

Synopsis: Current exception address.

Declaration: `function ExceptAddr : CodePointer`

Visibility: default

Description: ExceptAddr returns the address from the currently treated exception object when an exception is raised, and the stack is unwound.

See also: ExceptObject (1685), ExceptionErrorMessage (1685), ShowException (1751)

76.15.82 ExceptFrameCount

Synopsis: Number of frames included in an exception backtrace.

Declaration: `function ExceptFrameCount : LongInt`

Visibility: default

Description: `ExceptFrameCount` returns the number of frames that are included in an exception stack frame backtrace. The function returns 0 if there is currently no exception being handled. (i.e. it only makes sense to call this function in an `finally..end` or `except..end` block.

Errors: None.

See also: `ExceptFrames` (1685), `ExceptAddr` (1684), `ExceptObject` (1685), `#rtl.system.ExceptProc` (1345)

76.15.83 ExceptFrames

Synopsis: Return the current exception stack frames.

Declaration: `function ExceptFrames : PCodePointer`

Visibility: default

Description: `ExceptFrames` returns the current list frames on the exception stack. If there is no exception in progress, `Nil` is returned.

See also: `ExceptFrameCount` (1685), `ExceptAddr` (1684), `ExceptObject` (1685), `#rtl.system.ExceptProc` (1345)

76.15.84 ExceptionErrorMessage

Synopsis: Return a message describing the exception.

Declaration: `function ExceptionErrorMessage(ExceptObject: TObject;
 ExceptAddr: Pointer; Buffer: PAnsiChar;
 Size: Integer) : Integer`

Visibility: default

Description: `ExceptionErrorMessage` creates a string that describes the exception object `ExceptObject` at address `ExceptAddr`. It can be used to display exception messages. The string will be stored in the memory pointed to by `Buffer`, and will at most have `Size` characters.

The routine checks whether `ExceptObject` is a `Exception` (1821) object or not, and adapts the output accordingly.

See also: `ExceptObject` (1685), `ExceptAddr` (1684), `ShowException` (1751)

76.15.85 ExceptObject

Synopsis: Current Exception object.

Declaration: `function ExceptObject : TObject`

Visibility: default

Description: `ExceptObject` returns the currently treated exception object when an exception is raised, and the stack is unwound.

Errors: If there is no exception, the function returns `Nil`

See also: `ExceptAddr` (1684), `ExceptionErrorMessage` (1685), `ShowException` (1751)

76.15.86 Exchange

Declaration: `function Exchange<T>(var target: T; const newvalue: T) : T`

Visibility: default

76.15.87 ExcludeLeadingPathDelimiter

Synopsis: Strip the leading path delimiter of a path.

Declaration: `function ExcludeLeadingPathDelimiter(const Path: UnicodeString)
: UnicodeString
function ExcludeLeadingPathDelimiter(const Path: RawByteString)
: RawByteString`

Visibility: default

Description: `ExcludeLeadingPathDelimiter` will remove any path delimiter on the first position of `Path` if there is one. if there is none (or the path is empty), it is left untouched.

See also: `IncludeTrailingPathDelimiter` (1736), `IncludeLeadingPathDelimiter` (1736), `ExcludeTrailingPathDelimiter` (1687), `ConcatPaths` (1671)

Listing: `./examples/sysutex/ex95.pp`

Program Example95;

{ This program demonstrates the IncludeLeadingPathDelimiter function }

Uses sysutils;

Begin

// Will print "/this/path"

Writeln(IncludeLeadingPathDelimiter('this/path'));

// The same result

Writeln(IncludeLeadingPathDelimiter('/this/path'));

End.

76.15.88 ExcludeTrailingBackslash

Synopsis: Strip trailing directory separator from a pathname, if needed.

Declaration: `function ExcludeTrailingBackslash(const Path: UnicodeString)
: UnicodeString
function ExcludeTrailingBackslash(const Path: RawByteString)
: RawByteString`

Visibility: default

Description: `ExcludeTrailingBackslash` is provided for backwards compatibility with Delphi. Use `ExcludeTrailingPathDelimiter` (1687) instead.

See also: `IncludeTrailingPathDelimiter` (1736), `ExcludeTrailingPathDelimiter` (1687), `PathDelim` (1622), `IsPathDelimiter` (1741)

76.15.89 ExcludeTrailingPathDelimiter

Synopsis: Strip trailing directory separator from a pathname, if needed.

```
Declaration: function ExcludeTrailingPathDelimiter(const Path: UnicodeString)
                                          : UnicodeString
function ExcludeTrailingPathDelimiter(const Path: RawByteString)
                                          : RawByteString
```

Visibility: default

Description: `ExcludeTrailingPathDelimiter` removes the trailing path delimiter character (`PathDelimiter` (1622)) from `Path` if it is present, and returns the result.

See also: [ExcludeTrailingBackslash \(1686\)](#), [IncludeTrailingPathDelimiter \(1736\)](#), [PathDelim \(1622\)](#), [IsPathDelimiter \(1741\)](#)

76.15.90 ExecuteProcess

Synopsis: Execute another process (program).

```

Declaration: function ExecuteProcess(const Path: RawByteString;
                                     const ComLine: RawByteString;
                                     Flags: TExecuteFlags) : Integer
function ExecuteProcess(const Path: RawByteString;
                       const ComLine: Array of RawByteString;
                       Flags: TExecuteFlags) : Integer
function ExecuteProcess(const Path: UnicodeString;
                       const ComLine: UnicodeString;
                       Flags: TExecuteFlags) : Integer
function ExecuteProcess(const Path: UnicodeString;
                       const ComLine: Array of UnicodeString;
                       Flags: TExecuteFlags) : Integer

```

Visibility: default

Description: `ExecuteProcess` will execute the program in `Path`, passing it the arguments in `ComLine`. `ExecuteProcess` will then wait for the program to finish, and will return the exit code of the executed program. In case `ComLine` is a single string, it will be split out in an array of strings, taking into account common whitespace and quote rules.

The program specified in `Path` is not searched in the searchpath specified in the `PATH` environment variable, so the full path to the executable must be specified in `Path`, although some operating systems may perform this search anyway (notably, windows)

Flags can be used to control the passing of file handles: if `ExecInheritsHandles` is included, the file handles of the current process will be passed on to the newly executed process.

Errors: In case the program could not be executed or an other error occurs, an `EOSError` ([1818](#)) exception will be raised.

See also: [TExecuteFlags \(1631\)](#), [EOSError \(1818\)](#)

76.15.91 ExeSearch

Synopsis: Search for an executable.

Declaration: `function ExeSearch(const Name: UnicodeString;
 const DirList: UnicodeString) : UnicodeString
 function ExeSearch(const Name: RawByteString;
 const DirList: RawByteString) : RawByteString`

Visibility: default

Description: `ExeSearch` searches for an executable `Name` in the list of directories `DirList` (a list of directories, separator by `PathSeparator` (1359)). If the current OS also searches implicitly in the current working directory, the current directory is searched in the first place.

If the executable is found, then the full path of the executable is returned. If it is not found, an empty string is returned.

No check is performed whether the found file is actually executable.

See also: `FileSearch` (1702)

76.15.92 ExpandFileName

Synopsis: Expand a relative filename to an absolute filename.

Declaration: `function ExpandFileName(const FileName: UnicodeString) : UnicodeString
 function ExpandFileName(const FileName: RawByteString) : RawByteString`

Visibility: default

Description: `ExpandFileName` expands the filename to an absolute filename. It changes all directory separator characters to the one appropriate for the system first.

If an empty filename is passed, it is expanded to the current directory.

If `BasePath` is not passed, the current directory is used as base path.

Errors: None.

See also: `ExpandFileNameCase` (1689), `ExtractFileName` (1691), `ExtractFilePath` (1692), `ExtractFileDir` (1690), `ExtractFileDrive` (1691), `ExtractFileExt` (1691), `ExtractRelativePath` (1692)

Listing: `./examples/sysutex/ex33.pp`

Program Example33;

{ This program demonstrates the ExpandFileName function }

Uses sysutils;

Procedure Testit (F : **String**);

begin

Writeln (F, ' expands to : ', **ExpandFileName**(F));
end;

Begin

 Testit('ex33.pp');
 Testit(**ParamStr**(0));
 Testit('/pp/bin/win32/ppc386');
 Testit('\pp\bin\win32\ppc386');
 Testit('.');

End.

76.15.93 ExpandFileNameCase

Synopsis: Expand a filename entered as case insensitive to the full path as stored on the disk.

Declaration:

```
function ExpandFileNameCase(const FileName: UnicodeString;
                           out MatchFound: TFilenameCaseMatch)
                           : UnicodeString
function ExpandFileNameCase(const FileName: RawByteString;
                           out MatchFound: TFilenameCaseMatch)
                           : RawByteString
```

Visibility: default

Description: On case insensitive platforms, `ExpandFileNameCase` behaves similarly to `ExpandFileName` (1688) except for the fact that it returns the final part of the path with the same case of letters as found on the disk (if it exists - otherwise the case equals the one provided on input). On case sensitive platforms it also checks whether one or more full paths exist on disk which would correspond to the provided input if treated case insensitively and returns the first such match found and information whether the match is unique or not.

Note that the behaviour is basically undefined if the input includes wildcards characters. Normally, wildcards in the last part of path provided on input are resolved to the first corresponding item found on the disk, but it is better not to rely on that and use other more suitable functions if working with wildcards like `FindFirst` (1706)/`FindNext` (1708).

Errors: None.

See also: `ExpandFileName` (1688), `ExtractFileName` (1691), `ExtractFilePath` (1692), `ExtractFileDir` (1690), `ExtractFileDrive` (1691), `ExtractFileExt` (1691), `ExtractRelativePath` (1692)

Listing: ./examples/sysutex/ex33.pp

Program Example33;

{ This program demonstrates the ExpandFileName function }

Uses sysutils;

Procedure Testit (F : **String**);

begin

Writeln (F,' expands to : ',**ExpandFileName**(F));

end;

Begin

 Testit('ex33.pp');

 Testit(**ParamStr**(0));

 Testit('/pp/bin/win32/ppc386');

 Testit('\pp\bin\win32\ppc386');

 Testit('.');

End.

76.15.94 ExpandUNCFileName

Synopsis: Expand a relative filename to an absolute UNC filename.

Declaration:

```
function ExpandUNCFileName(const FileName: UnicodeString)
                           : UnicodeString
```

```
function ExpandUNCFileName(const FileName: RawByteString)
    : RawByteString
```

Visibility: default

Description: `ExpandUNCFileName` runs `ExpandFileName` (1688) on `FileName` and then attempts to replace the drive letter by the name of a shared disk.

Errors: If an unexpected error occurs while determining the name of the shared disk, an empty string is returned.

See also: `ExpandFileName` (1688), `ExtractFileName` (1691), `ExtractFilePath` (1692), `ExtractFileDir` (1690), `ExtractFileDrive` (1691), `ExtractFileExt` (1691), `ExtractRelativePath` (1692)

76.15.95 ExtractFileDir

Synopsis: Extract the drive and directory part of a filename.

Declaration: `function ExtractFileDir(const FileName: UnicodeString) : UnicodeString`
`function ExtractFileDir(const FileName: RawByteString) : RawByteString`

Visibility: default

Description: `ExtractFileDir` returns only the directory part of `FileName`, including a drive letter. The directory name has NO ending directory separator, in difference with `ExtractFilePath` (1692).

Errors: None.

See also: `ExtractFileName` (1691), `ExtractFilePath` (1692), `ExtractFileDir` (1690), `ExtractFileDrive` (1691), `ExtractFileExt` (1691), `ExtractRelativePath` (1692)

Listing: ./examples/sysutex/ex34.pp

Program Example34;

```
{ This program demonstrates the ExtractFileName function }
{$H+}
Uses sysutils;
```

Procedure Testit(F : String);

```
begin
  Writeln ('FileName : ',F);
  Writeln ('Has Name : ',ExtractFileName(F));
  Writeln ('Has Path : ',ExtractFilePath(F));
  Writeln ('Has Extension : ',ExtractFileExt(F));
  Writeln ('Has Directory : ',ExtractFileDir(F));
  Writeln ('Has Drive : ',ExtractFileDrive(F));
end;
```

```
Begin
  Testit (Paramstr(0));
  Testit ('/usr/local/bin/mysql');
  Testit ('c:\pp\bin\win32\ppc386.exe');
  Testit ('/pp/bin/win32/ppc386.exe');
End.
```

76.15.96 ExtractFileDrive

Synopsis: Extract the drive part from a filename.

Declaration: `function ExtractFileDrive(const FileName: UnicodeString) : UnicodeString`
`function ExtractFileDrive(const FileName: RawByteString) : RawByteString`

Visibility: default

Description: `ExtractFileDrive` extracts the drive letter from a filename. Note that some operating systems do not support drive letters.

For an example, see `ExtractFileDir` (1690).

See also: `ExtractFileName` (1691), `ExtractFilePath` (1692), `ExtractFileDir` (1690), `ExtractFileDrive` (1691), `ExtractFileExt` (1691), `ExtractRelativePath` (1692)

76.15.97 ExtractFileExt

Synopsis: Return the extension from a filename.

Declaration: `function ExtractFileExt(const FileName: UnicodeString) : UnicodeString`
`function ExtractFileExt(const FileName: RawByteString) : RawByteString`

Visibility: default

Description: `ExtractFileExt` returns the extension (including the . (dot) character) of `FileName`.

For an example, see `ExtractFileDir` (1690).

Errors: None.

See also: `ChangeFileExt` (1666), `ExtractFileName` (1691), `ExtractFilePath` (1692), `ExtractFileDir` (1690), `ExtractFileDrive` (1691), `ExtractFileExt` (1691), `ExtractRelativePath` (1692)

76.15.98 ExtractFileName

Synopsis: Extract the filename part from a full path filename.

Declaration: `function ExtractFileName(const FileName: UnicodeString) : UnicodeString`
`function ExtractFileName(const FileName: RawByteString) : RawByteString`

Visibility: default

Description: `ExtractFileName` returns the filename part from `FileName`. The filename consists of all characters after the last directory separator character ('/' or '\') or drive letter.

The full filename can always be reconstructed by concatenating the result of `ExtractFilePath` (1692) and `ExtractFileName`.

For an example, see `ExtractFileDir` (1690).

Errors: None.

See also: `ExtractFileName` (1691), `ExtractFilePath` (1692), `ExtractFileDir` (1690), `ExtractFileDrive` (1691), `ExtractFileExt` (1691), `ExtractRelativePath` (1692)

76.15.99 ExtractFilePath

Synopsis: Extract the path from a filename.

Declaration: `function ExtractFilePath(const FileName: UnicodeString) : UnicodeString`
`function ExtractFilePath(const FileName: RawByteString) : RawByteString`

Visibility: default

Description: `ExtractFilePath` returns the path part (including drive letter) from `FileName`. The path consists of all characters before the last directory separator character ('/' or '\'), including the directory separator itself. In case there is only a drive letter, that will be returned.

The full filename can always be reconstructed by concatenating the result of `ExtractFilePath` and `ExtractFileName` (1691).

For an example, see `ExtractFileDir` (1690).

Errors: None.

See also: `ExtractFileName` (1691), `ExtractFilePath` (1692), `ExtractFileDir` (1690), `ExtractFileDrive` (1691), `ExtractFileExt` (1691), `ExtractRelativePath` (1692)

76.15.100 ExtractRelativePath

Synopsis: Extract a relative path from a filename, given a base directory.

Declaration: `function ExtractRelativePath(const BaseName: UnicodeString;`
`const DestName: UnicodeString)`
`: UnicodeString`
`function ExtractRelativePath(const BaseName: RawByteString;`
`const DestName: RawByteString)`
`: RawByteString`

Visibility: default

Description: `ExtractRelativePath` constructs a relative path to go from `BaseName` to `DestName`. If `DestName` is on another drive (Not on Unix-like platforms) then the whole `Destname` is returned.

Note that directories must end on a path delimiter for this function to work correctly. If not, the last part is stripped and treated as a file name.

Errors: None.

See also: `ExtractFileName` (1691), `ExtractFilePath` (1692), `ExtractFileDir` (1690), `ExtractFileDrive` (1691), `ExtractFileExt` (1691)

Listing: ./examples/sysutex/ex35.pp

Program Example35;

{ This program demonstrates the ExtractRelativePath function }

Uses sysutils;

Procedure Testit (FromDir, ToDir : **String**);

begin

Write ('From "', FromDir, '" to "', ToDir, '" via "');

WriteLn (ExtractRelativePath(FromDir, ToDir), "");

end;

Begin

```

Testit ('/pp/src/compiler/', '/pp/bin/win32/ppc386/');
Testit ('/pp/bin/win32/ppc386/', '/pp/src/compiler/');
Testit ('/pp/bin/win32/', '/pp/src/compiler/ppcx386/');
Testit ('/pp/bin/win32/', '/pp/src/compiler/ppcx386/');
Testit ('/pp/bin/win32/', '/pp/src/compiler/ppcx386/');
Testit ('e:/pp/bin/win32/ppc386/', 'd:/pp/src/compiler/');
Testit ('e:/pp/bin/win32/ppc386/', 'd:/pp/src/compiler/');
Testit ('C:\FPC\3.0.2\', 'C:\FPC\3.0.2\');
Testit ('C:\FPC\3.0.2\', 'C:\FPC\3.0.4rc1\');
Testit ('Q:\', 'Q:\FPC\3.0.4rc1\');

```

End.**76.15.101 ExtractShortPathName**

Synopsis: Returns a 8.3 path name.

Declaration: `function ExtractShortPathName(const FileName: UnicodeString) : UnicodeString`
`function ExtractShortPathName(const FileName: RawByteString) : RawByteString`

Visibility: default

Description: `ExtractShortPathName` returns a 8.3 compliant filename that represents the same file as `FileName`. On platforms other than windows, this is `FileName` itself.

See also: `ExtractFilePath` ([1692](#)), `ExtractFileName` ([1691](#))

76.15.102 FileAge

Synopsis: Return the timestamp of a file.

Declaration: `function FileAge(const FileName: UnicodeString) : Int64`
`function FileAge(const FileName: UnicodeString; out FileDateTime: TDateTime; FollowLink: Boolean) : Boolean`
`function FileAge(const FileName: RawByteString; out FileDateTime: TDateTime; FollowLink: Boolean) : Boolean`
`function FileAge(const FileName: RawByteString) : Int64`

Visibility: default

Description: `FileAge` in it's simplest form returns the last modification time of file `FileName` as an opaque os-dependent file timestamp. In this case, the return value can be transformed to a `TDateTime` value with the `FileDateToDateTime` ([1696](#)) function.

In case a `FileDateTime` argument is supplied, the `FileAge` function will do the transformation by itself, and the file time will be returned in the `FileDateTime` argument as a `TDateTime` value. The function return value is then a boolean indicating success or failure.

The `FollowSymLink` parameter can be set to `False` to read the timestamp of a symbolic link, instead of the timestamp of the file the symbolic link points to (which is the default behaviour).

`FileAge` cannot be used on directories, it will return -1 (or `False`) if `FileName` indicates a directory.

Errors: In case of errors, `-1` or `False` is returned.

See also: [FileDateToDateTime \(1696\)](#), [FileExists \(1697\)](#), [FileGetAttr \(1698\)](#)

Listing: `./examples/sysutex/ex36.pp`

Program Example36;

{ This program demonstrates the FileAge function }

Uses sysutils;

Var S : TDateTime;

fa : Longint;

Begin

fa:=FileAge('ex36.pp');

If Fa<>-1 **then**

begin

S:=FileDateToDateTime(fa);

WriteLn ('I'm from ',DateTimeToStr(S))

end;

End.

76.15.103 FileAgeUTC

Declaration:

```
function FileAgeUTC(const FileName: UnicodeString;
                    out FileDateTimeUTC: TDateTime; FollowLink: Boolean)
    : Boolean
function FileAgeUTC(const FileName: RawByteString;
                    out FileDateTimeUTC: TDateTime; FollowLink: Boolean)
    : Boolean
```

Visibility: default

76.15.104 FileClose

Synopsis: Close a file handle.

Declaration: `procedure FileClose(Handle: THandle)`

Visibility: default

Description: `FileClose` closes the file handle `Handle`. After this call, attempting to read or write from the handle will result in an error.

For an example, see [FileCreate \(1694\)](#)

Errors: None.

See also: [FileCreate \(1694\)](#), [FileWrite \(1705\)](#), [FileOpen \(1700\)](#), [FileRead \(1701\)](#), [FileTruncate \(1705\)](#), [FileSeek \(1703\)](#)

76.15.105 FileCreate

Synopsis: Create a new file and return a handle to it.

Declaration:

```

function FileCreate(const FileName: UnicodeString) : THandle
function FileCreate(const FileName: UnicodeString; Rights: Integer)
    : THandle
function FileCreate(const FileName: UnicodeString; ShareMode: Integer;
    Rights: Integer) : THandle
function FileCreate(const FileName: RawByteString) : THandle
function FileCreate(const FileName: RawByteString; Rights: Integer)
    : THandle
function FileCreate(const FileName: RawByteString; ShareMode: Integer;
    Rights: Integer) : THandle

```

Visibility: default

Description: `FileCreate` creates a new file with name `FileName` on the disk and returns a file handle which can be used to read or write from the file with the `FileRead` (1701) and `FileWrite` (1705) functions.

If a file with name `FileName` already existed on the disk, it is overwritten.

The `ShareMode` parameter determines how file sharing is handled for the new file. It can have one or more of the following values, OR-ed together:

Table 76.25:

Name	Description
<code>fmShareCompat</code>	Open file in DOS share-compatibility mode.
<code>fmShareDenyNone</code>	Do not lock file.
<code>fmShareDenyRead</code>	Lock file so other processes cannot read.
<code>fmShareDenyWrite</code>	Lock file so other processes can only read.
<code>fmShareExclusive</code>	Lock file for exclusive use.

The optional `Rights` parameter only has an effect under UNIX, where it can be used to set the mode (read, write, execute, sticky bit, setgid and setuid flags) of the created file to the specified custom value. On other platforms, the `Rights` parameter is ignored.

Errors: If an error occurs (e.g. disk full or non-existent path), the function returns `THandle(-1)`.

See also: `FileClose` (1694), `FileWrite` (1705), `FileOpen` (1700), `FileRead` (1701), `FileTruncate` (1705), `FileSeek` (1703)

Listing: `./examples/sysutex/ex37.pp`

Program Example37;

{ This program demonstrates the FileCreate/FileSeek/FileReed/FileTruncate functions }

Uses sysutils;

Var I,J,F : Longint;

Begin

```

F:=FileCreate('test.dat');
If F=-1 then
    Halt(1);
For I:=0 to 100 do
    FileWrite(F,I,SizeOf(i));
FileClose(f);

```

```

F:=FileOpen ('test.dat',fmOpenRead);
For I:=0 to 100 do
  begin
    FileRead (F,J,SizeOf(J));
    If J<>I then
      Writeln ('Mismatch at file position ',I)
    end;
  FileSeek(F,0,fsFromBeginning);
  Randomize;
  Repeat
    FileSeek(F,Random(100)*4,fsFromBeginning);
    FileRead (F,J,SizeOf(J));
    Writeln ('Random read : ',j);
  Until J>80;
  FileClose(F);
  F:=FileOpen('test.dat',fmOpenWrite);
  I:=50*SizeOf(Longint);
  If FileTruncate(F,I) then
    Writeln('Successfully truncated file to ',I,' bytes. ');
  FileClose(F);
End.

```

76.15.106 FileDateToDateTime

Synopsis: Convert a `FileDate` value to a `TDateTime` value.

Declaration: `function FileDateToDateTime (Filedate: Int64) : TDateTime`

Visibility: default

Description: `FileDateToDateTime` converts the date/time encoded in `filedate` to a `TDateTime` encoded form. It can be used to convert date/time values returned by the `FileAge` ([1693](#)) or `FindFirst` ([1706](#))/`FindNext` ([1708](#)) functions to `TDateTime` form.

Errors: None.

See also: `DateTimeToFileDate` ([1674](#))

Listing: `./examples/sysutex/ex13.pp`

Program Example13;

{ This program demonstrates the FileDateToDateTime function }

Uses sysutils;

Var

 ThisAge : Longint;

Begin

 Write ('ex13.pp created on :');

 ThisAge:=FileAge('ex13.pp');

 Writeln (DateTimeToStr(FileDateToDateTime(ThisAge)));

End.

76.15.107 FileDateToUniversal

Declaration: `function FileDateToUniversal(Filedate: Int64) : TDateTime`

Visibility: default

76.15.108 FileExists

Synopsis: Check whether a particular file exists in the file system.

Declaration: `function FileExists(const FileName: UnicodeString; FollowLink: Boolean)
: Boolean
function FileExists(const FileName: RawByteString; FollowLink: Boolean)
: Boolean`

Visibility: default

Description: `FileExists` returns `True` if a file with name `FileName` exists on the disk, `False` otherwise.

On windows, this function will return `False` if a directory is passed as `FileName`.

On Unixes, passing a directory name used to result in `True`. (The rationale is that on UNIX, a directory is a file as well). As of version 3.2.0, this behaviour has been changed for a Delphi-compatible approach.

Note that this function accepts a single filename as an argument, without wildcards. To check for the existence of multiple files, see the `FindFirst` (1706) function.

Errors: None.

See also: `FindFirst` (1706), `FileAge` (1693), `FileGetAttr` (1698), `FileSetAttr` (1704)

Listing: `./examples/sysutex/ex38.pp`

Program Example38;

{ This program demonstrates the FileExists function }

Uses sysutils;

Begin

If `FileExists(ParamStr(0))` **Then**

Writeln ('All is well, I seem to exist.');

End.

76.15.109 FileFlush

Synopsis: Synchronize file contents to disk.

Declaration: `function FileFlush(Handle: THandle) : Boolean`

Visibility: default

Description: `FileFlush` can be used to force the operating system to write any cached changes to file `Handle` to disk. It returns `True` if the operation completed successfully. Note that not all operating systems support this operation.

Errors: On error, check the `GetLastError` (1731) function result for more information about the error.

See also: `FileClose` (1694), `FileWrite` (1705), `FileOpen` (1700)

76.15.110 FileGetAttr

Synopsis: Return attributes of a file.

Declaration: `function FileGetAttr(const FileName: UnicodeString) : LongInt`
`function FileGetAttr(const FileName: RawByteString) : LongInt`

Visibility: default

Description: `FileGetAttr` returns the attribute settings of file `FileName`. The attribute is a OR-ed combination of the following constants:

faReadOnly The file is read-only.

faHidden The file is hidden. (On UNIX, this means that the filename starts with a dot)

faSysFile The file is a system file (On UNIX, this means that the file is a character, block or FIFO file).

faVolumeId Volume Label. Only for DOS/Windows on a plain FAT (not VFAT or Fat32) file system.

faDirectory File is a directory.

faArchive file should be archived. Not possible on Unix

Errors: In case of error, -1 is returned.

See also: `FileSetAttr` ([1704](#)), `FileAge` ([1693](#)), `FileGetDate` ([1699](#))

Listing: `./examples/sysutex/ex40.pp`

Program Example40;

{ This program demonstrates the FileGetAttr function }

Uses sysutils;

Procedure Testit (**Name** : **String**);

Var F : Longint;

Begin

F:=FileGetAttr(Name);

If F<>-1 then

begin

Writeln ('Testing : ',Name);

If (F and faReadOnly)<>0 then

Writeln ('File is ReadOnly');

If (F and faHidden)<>0 then

Writeln ('File is hidden');

If (F and faSysFile)<>0 then

Writeln ('File is a system file');

If (F and faVolumeId)<>0 then

Writeln ('File is a disk label');

If (F and faArchive)<>0 then

Writeln ('File is archive file');

If (F and faDirectory)<>0 then

Writeln ('File is a directory');

end

else

Writeln ('Error reading attributes of ',Name);

end;

```

begin
  testit ('ex40.pp');
  testit (ParamStr(0));
  testit ('.');
  testit ('/');
End.

```

76.15.111 FileGetDate

Synopsis: Return the file time of an opened file.

Declaration: `function FileGetDate(Handle: THandle) : Int64`
`function FileGetDate(Handle: THandle; out FileDateTime: TDateTime)`
`: Boolean`

Visibility: default

Description: `FileGetdate` returns the filetype of the opened file with file handle `Handle`. It is the same as `FileAge` ([1693](#)), with this difference that `FileAge` only needs the file name, while `FileGetDate` needs an open file handle.

Errors: On error, -1 is returned.

See also: `FileAge` ([1693](#))

Listing: `./examples/sysutex/ex39.pp`

Program Example39;

{ This program demonstrates the FileGetDate function }

Uses sysutils;

Var F,D : Longint;

Begin

F:=FileCreate('test.dat');

D:=FileGetDate(F);

WriteLn ('File created on ',DateTimeToStr(FileDateTime(D)));

FileClose(F);

DeleteFile('test.dat');

End.

76.15.112 FileGetDateTimeInfo

Declaration: `function FileGetDateTimeInfo(const FileName: string;`
`out DateTime: TDateTimeInfoRec;`
`FollowLink: Boolean) : Boolean`

Visibility: default

76.15.113 FileGetDateUTC

Declaration: `function FileGetDateUTC(Handle: THandle; out FileDateTimeUTC: TDateTime)
: Boolean`

Visibility: default

76.15.114 FileGetSymLinkTarget

Synopsis: Get the target filename and size of a symlink.

Declaration: `function FileGetSymLinkTarget(const FileName: UnicodeString;
out SymLinkRec: TUnicodeSymLinkRec)
: Boolean
function FileGetSymLinkTarget(const FileName: UnicodeString;
out TargetName: UnicodeString) : Boolean
function FileGetSymLinkTarget(const FileName: RawByteString;
out SymLinkRec: TRawbyteSymLinkRec)
: Boolean
function FileGetSymLinkTarget(const FileName: RawByteString;
out TargetName: RawByteString) : Boolean`

Visibility: default

Description: `FileGetSymLinkTarget` returns the target filename, size and attributes of a symbolic link in `SymLinkRec`. On unix-like systems, the filemode is also returned. The filename can be a relative filename, it will not be expanded.

See also: `TUnicodeSymLinkRec` ([1810](#)), `TRawbyteSymLinkRec` ([1808](#))

76.15.115 FileIsReadOnly

Synopsis: Check whether a file is read-only.

Declaration: `function FileIsReadOnly(const FileName: UnicodeString) : Boolean
function FileIsReadOnly(const FileName: RawByteString) : Boolean`

Visibility: default

Description: `FileIsReadOnly` checks whether `FileName` exists in the file system and is a read-only file. If this is the case, the function returns `True`, otherwise `False` is returned.

See also: `FileExists` ([1697](#))

76.15.116 FileOpen

Synopsis: Open an existing file and return a file handle.

Declaration: `function FileOpen(const FileName: UnicodeString; Mode: Integer)
: THandle
function FileOpen(const FileName: RawByteString; Mode: Integer)
: THandle`

Visibility: default

Description: `FileOpen` opens a file with name `FileName` with mode `Mode`. `Mode` can be one of the following constants:

fmOpenReadOpen file in read-only mode.

fmOpenWriteOpen file in write-only mode.

fmOpenReadWriteOpen file in read/write mode.

Under Windows and Unix, the above mode can be or-ed with one of the following sharing/locking flags:

fmShareCompatOpen file in DOS share-compatibility mode.

fmShareExclusiveLock file for exclusive use.

fmShareDenyWriteLock file so other processes can only read.

fmShareDenyReadLock file so other processes cannot read.

fmShareDenyNoneDo not lock file.

If the file has been successfully opened, it can be read from or written to (depending on the `Mode` parameter) with the `FileRead` (1701) and `FileWrite` functions.

Remark Remark that you cannot open a file if it doesn't exist yet, i.e. it will not be created for you. If you want to create a new file, or overwrite an old one, use the `FileCreate` (1694) function.

There are some limitations to the sharing modes.

- 1.Sharing modes are only available on Unix and Windows platforms.
- 2.Unix only support sharing modes as of 2.4.0.
- 3.`fmShareDenyRead` only works under Windows at this time, and will always result in an error on Unix platforms because its file locking APIs do not support this concept.
- 4.File locking is advisory on Unix platforms. This means that the locks are only checked when a file is opened using a file locking mode. In other cases, existing locks are simply ignored. In particular, this means that `fmShareDenyNone` has no effect under Unix, because this can only be implemented as “use no locking” on those platforms. As a result, opening a file using this mode will always succeed under Unix as far as the locking is concerned, even if the file has already been opened using `fmShareExclusive`.
- 5.Under Solaris, closing a single file handle associated with a file will result in all locks on that file (even via other handles) being destroyed due to the behaviour of the underlying API (`fcntl`). Because of the same reason, on Solaris you cannot use `fmShareDenyWrite` in combination with `fmOpenWrite`, nor `fmShareExclusive` in combination with `fmOpenRead` although both work with `fmOpenReadWrite`.

For an example, see `FileCreate` (1694)

Errors: On Error, `THandle (-1)` is returned.

See also: `fmOpenRead` (1619), `fmOpenWrite` (1619), `fmOpenReadWrite` (1619), `fmShareDenyWrite` (1620), `fmShareExclusive` (1620), `fmShareDenyRead` (1620), `fmShareDenyNone` (1619), `fmShareCompat` (1619), `FileClose` (1694), `FileWrite` (1705), `FileCreate` (1694), `FileRead` (1701), `FileTruncate` (1705), `FileSeek` (1703)

76.15.117 FileRead

Synopsis: Read data from a file handle in a buffer.

Declaration: `function FileRead(Handle: THandle; out Buffer; Count: LongInt) : LongInt`

Visibility: default

Description: `FileRead` reads `Count` bytes from file-handle `Handle` and stores them into `Buffer`. `Buffer` must be at least `Count` bytes long. No checking on this is performed, so be careful not to overwrite any memory. `Handle` must be the result of a `FileOpen` (1700) call.

The function returns the number of bytes actually read, or -1 on error.

For an example, see `FileCreate` (1694)

Errors: On error, -1 is returned.

See also: `FileClose` (1694), `FileWrite` (1705), `FileCreate` (1694), `FileOpen` (1700), `FileTruncate` (1705), `FileSeek` (1703)

76.15.118 FileSearch

Synopsis: Search for a file in a path.

Declaration:

```
function FileSearch(const Name: UnicodeString;
                   const DirList: UnicodeString;
                   Options: TFileSearchOptions) : UnicodeString
function FileSearch(const Name: UnicodeString;
                   const DirList: UnicodeString;
                   ImplicitCurrentDir: Boolean) : UnicodeString
function FileSearch(const Name: RawByteString;
                   const DirList: RawByteString;
                   Options: TFileSearchOptions) : RawByteString
function FileSearch(const Name: RawByteString;
                   const DirList: RawByteString;
                   ImplicitCurrentDir: Boolean) : RawByteString
```

Visibility: default

Description: `FileSearch` looks for the file `Name` in `DirList`, where `dirlist` is a list of directories, separated by semicolons or colons. It returns the full filename of the first match found. The optional `Options` parameter may be specified to influence the behaviour of the search algorithm. It is a set of the following options:

sfoImplicitCurrentDir Always search the current directory first, even if it is not specified.

sfoStripQuotes Strip quotes from the components in the search path.

A deprecated form of the function allowed to specify using the boolean `ImplicitCurrentDir` parameter whether the current directory was searched implicitly or not. By default, the current directory is searched.

Errors: On error, an empty string is returned.

See also: `ExpandFileName` (1688), `FindFirst` (1706)

Listing: `./examples/sysutex/ex41.pp`

Program Example41;

{ Program to demonstrate the FileSearch function. }

Uses Sysutils;

Const
{ \$ifdef unix }

```

    FN = 'find';
    P = './bin:/usr/bin';
  {$else}
    FN = 'find.exe';
    P = 'c:\dos;c:\windows;c:\windows\system;c:\windows\system32';
  {$endif}

begin
  Writeln ('find is in : ',FileSearch (FN,P));
end.

```

76.15.119 FileSeek

Synopsis: Set the current file position on a file handle.

Declaration: `function FileSeek(Handle: THandle; FOffset: LongInt; Origin: LongInt) : LongInt`
`function FileSeek(Handle: THandle; FOffset: Int64; Origin: LongInt) : Int64`

Visibility: default

Description: `FileSeek` sets the file pointer on position `Offset`, starting from `Origin`. `Origin` can be one of the following values:

fsFromBeginning `Offset` is relative to the first byte of the file. This position is zero-based. i.e. the first byte is at offset 0.

fsFromCurrent `Offset` is relative to the current position.

fsFromEnd `Offset` is relative to the end of the file. This means that `Offset` can only be zero or negative in this case.

If successful, the function returns the new file position, relative to the beginning of the file.

Remark The above mentioned constants do not exist in Delphi.

Errors: On error, -1 is returned.

See also: `FileClose` ([1694](#)), `FileWrite` ([1705](#)), `FileCreate` ([1694](#)), `FileOpen` ([1700](#)), `FileRead` ([1701](#)), `FileTruncate` ([1705](#))

Listing: `./examples/sysutex/ex37.pp`

Program Example37;

{ This program demonstrates the FileCreate/FileSeek/FileReed/FileTruncate functions }

Uses sysutils;

Var I,J,F : Longint;

Begin

```

F:=FileCreate ('test.dat');
If F=-1 then
  Halt(1);
For I:=0 to 100 do
  FileWrite(F,I,SizeOf(i));
FileClose(f);

```

```

F:=FileOpen ('test.dat',fmOpenRead);
For I:=0 to 100 do
  begin
    FileRead (F,J,SizeOf(J));
    If J<>I then
      Writeln ('Mismatch at file position ',I)
    end;
  FileSeek(F,0,fsFromBeginning);
  Randomize;
  Repeat
    FileSeek(F,Random(100)*4,fsFromBeginning);
    FileRead (F,J,SizeOf(J));
    Writeln ('Random read : ',j);
  Until J>80;
  FileClose(F);
  F:=FileOpen('test.dat',fmOpenWrite);
  I:=50*SizeOf(Longint);
  If FileTruncate(F,I) then
    Writeln('Successfully truncated file to ',I,' bytes. ');
  FileClose(F);
End.

```

76.15.120 FileSetAttr

Synopsis: Set the attributes of a file.

Declaration: `function FileSetAttr(const Filename: UnicodeString; Attr: LongInt) : LongInt`
`function FileSetAttr(const Filename: RawByteString; Attr: LongInt) : LongInt`

Visibility: default

Description: `FileSetAttr` sets the attributes of `FileName` to `Attr`. If the function was successful, 0 is returned, -1 otherwise. `Attr` can be set to an OR-ed combination of the predefined `faXXX` constants.

This function is not implemented on Unixes.

Errors: On error, -1 is returned (always on Unixes).

See also: `FileGetAttr` ([1698](#)), `FileGetDate` ([1699](#)), `FileSetDate` ([1704](#))

76.15.121 FileSetDate

Synopsis: Set the date of a file.

Declaration: `function FileSetDate(const FileName: UnicodeString; Age: Int64) : LongInt`
`function FileSetDate(const FileName: UnicodeString; const FileDateTime: TDateTime) : LongInt`
`function FileSetDate(const FileName: RawByteString; Age: Int64) : LongInt`
`function FileSetDate(const FileName: RawByteString; const FileDateTime: TDateTime) : LongInt`
`function FileSetDate(Handle: THandle; Age: Int64) : LongInt`
`function FileSetDate(Handle: THandle; const FileDateTime: TDateTime) : LongInt`

Visibility: default

Description: `FileSetDate` sets the file date of the open file with handle `Handle` or to `Age`, where `Age` is a DOS date-and-time stamp value.

Alternatively, the filename may be specified with the `FileName` argument. This variant of the call is mandatory on Unixes, since there is no OS support for setting a file timestamp based on a handle. (the handle may not be a real file at all).

The function returns zero if successful.

Errors: On Unix, the handle variant always returns -1, since this is impossible to implement. On Windows and DOS, a negative error code is returned.

76.15.122 FileSetDateUTC

Declaration:

```
function FileSetDateUTC(const FileName: UnicodeString;
                        const FileDateTimeUTC: TDateTime) : LongInt
function FileSetDateUTC(const FileName: RawByteString;
                        const FileDateTimeUTC: TDateTime) : LongInt
function FileSetDateUTC(Handle: THandle;
                        const FileDateTimeUTC: TDateTime) : LongInt
```

Visibility: default

76.15.123 FileTruncate

Synopsis: Truncate an open file to a given size.

Declaration: `function FileTruncate(Handle: THandle; Size: Int64) : Boolean`

Visibility: default

Description: `FileTruncate` truncates the file with handle `Handle` to `Size` bytes. The file must have been opened for writing prior to this call. The function returns `True` is successful, `False` otherwise.

For an example, see `FileCreate` ([1694](#)).

Errors: On error, the function returns `False`.

See also: `FileClose` ([1694](#)), `FileWrite` ([1705](#)), `FileCreate` ([1694](#)), `FileOpen` ([1700](#)), `FileRead` ([1701](#)), `FileSeek` ([1703](#))

76.15.124 FileWrite

Synopsis: Write data from a buffer to a given file handle.

Declaration: `function FileWrite(Handle: THandle; const Buffer; Count: LongInt) : LongInt`

Visibility: default

Description: `FileWrite` writes `Count` bytes from `Buffer` to the file with handle `Handle`. Prior to this call, the file must have been opened for writing. `Buffer` must be at least `Count` bytes large, or a memory access error may occur.

The function returns the number of bytes written, or -1 in case of an error.

For an example, see `FileCreate` ([1694](#)).

Errors: In case of error, -1 is returned.

See also: FileClose ([1694](#)), FileCreate ([1694](#)), FileOpen ([1700](#)), FileRead ([1701](#)), FileTruncate ([1705](#)), FileSeek ([1703](#))

76.15.125 FindClose

Synopsis: Close a find handle.

Declaration: `procedure FindClose(var F: TUnicodeSearchRec)`
`procedure FindClose(var F: TRawbyteSearchRec)`

Visibility: default

Description: FindClose ends a series of FindFirst ([1706](#))/FindNext ([1708](#)) calls, and frees any memory used by these calls. It is *absolutely* necessary to do this call, or huge memory losses may occur.

For an example, see FindFirst ([1706](#)).

Errors: None.

See also: FindFirst ([1706](#)), FindNext ([1708](#))

76.15.126 FindCmdLineSwitch

Synopsis: Check whether a certain switch is present on the command-line.

Declaration: `function FindCmdLineSwitch(const Switch: string;`
`const Chars: TSysCharSet; IgnoreCase: Boolean)`
`: Boolean`
`function FindCmdLineSwitch(const Switch: string; IgnoreCase: Boolean)`
`: Boolean`
`function FindCmdLineSwitch(const Switch: string) : Boolean`
`function FindCmdLineSwitch(const Switch: string; var Value: string;`
`IgnoreCase: Boolean;`
`const SwitchTypes: TCmdLineSwitchTypes)`
`: Boolean`

Visibility: default

Description: FindCmdLineSwitch will check all command-line arguments for the presence of the option Switch. It will return True if it was found, False otherwise. Characters that appear in Chars (default is SwitchChars ([1624](#))) are assumed to indicate an option (switch). If the parameter IgnoreCase is True, case will be ignored when looking for the switch. Default is to search case sensitive.

Errors: None.

See also: SwitchChars ([1624](#))

76.15.127 FindFirst

Synopsis: Start a file search and return a findhandle.

Declaration: `function FindFirst(const Path: UnicodeString; Attr: LongInt;`
`out Rslt: TUnicodeSearchRec) : LongInt`
`function FindFirst(const Path: RawByteString; Attr: LongInt;`
`out Rslt: TRawbyteSearchRec) : LongInt`

Visibility: default

Description: `FindFirst` looks for files that match the name (possibly with wildcards) in `Path` and extra attributes `Attr`. It then fills up the `Rslt` record with data gathered about the file. It returns 0 if a file matching the specified criteria is found, a nonzero value (-1 on Unix-like platforms) otherwise.

`Attr` is an or-ed combination of the following constants:

faAnyFile Find any file (this is a combination of the other flags except `faSymlink`).

faReadOnly The file is read-only.

faHidden The file is hidden. (On UNIX, this means that the filename starts with a dot)

faSysFile The file is a system file (On unix, this means that the file is a character, block or FIFO file).

faVolumeId Drive volume Label. Not possible under unix, and on Windows-like systems, this works only for plan FAT (not Fat32 or VFAT) file systems.

faDirectory File is a directory.

faArchive file needs to be archived. Not possible on Unix

faSymlink Report symlinks instead of following them

It is a common misconception that `Attr` specifies a set of attributes which must be matched in order for a file to be included in the list. This is not so: The value of `Attr` specifies *additional* attributes, this means that the returned files are either normal files or have an attribute which is present in `Attr`.

Specifically: specifying `faDirectory` as a value for `Attr` does not mean that only directories will be returned. Normal files *and* directories will be returned.

Since `faSymlink` is not included in `faAnyFile`, to find all possible files and symlinks you need to call `FindFirst` so:

```
if FindFirst('*', faAnyFile or faSymlink, info)=0 then
  // do your thing
```

The `Rslt` record can be fed to subsequent calls to `FindNext`, in order to find other files matching the specifications.

Remark A successful `FindFirst` call must *always* be followed by a `FindClose` (1706) call with the same `Rslt` record. Failure to do so will result in memory leaks. If the `findfirst` call failed (i.e. returned a nonzero handle) there is no need to call `FindClose`.

Errors: On error the function returns -1 on Unix-like platforms, a nonzero error code on Windows.

See also: `FindClose` (1706), `FindNext` (1708)

Listing: `./examples/sysutex/ex43.pp`

Program Example43;

{ This program demonstrates the FindFirst function }

Uses SysUtils;

Var Info : TSearchRec;
Count : Longint;

Begin

Count:=0;

If FindFirst ('*',faAnyFile,Info)=0 **then**
begin

```

Repeat
  Inc(Count);
  With Info do
    begin
      If (Attr and faDirectory) = faDirectory then
        Write('Dir : ');
        Writeln (Name:40,Size:15);
      end;
    Until FindNext(info)<>0;
    FindClose(Info);
  end;
  Writeln ('Finished search. Found ',Count,' matches');
End.

```

76.15.128 FindNext

Synopsis: Find the next entry in a findhandle.

Declaration: `function FindNext (var Rslt: TUnicodeSearchRec) : LongInt`
`function FindNext (var Rslt: TRawbyteSearchRec) : LongInt`

Visibility: default

Description: `FindNext` finds a next occurrence of a search sequence initiated by `FindFirst`. If another record matching the criteria in `Rslt` is found, 0 is returned, a nonzero constant is returned otherwise.

Remark The last `FindNext` call must *always* be followed by a `FindClose` call with the same `Rslt` record. Failure to do so will result in memory loss.

For an example, see `FindFirst` ([1706](#))

Errors: On error (no more file is found), a nonzero constant is returned.

See also: `FindFirst` ([1706](#)), `FindClose` ([1706](#))

76.15.129 FloatToCurr

Synopsis: Convert a float to a Currency value.

Declaration: `function FloatToCurr (const Value: Extended) : Currency`

Visibility: default

Description: `FloatToCurr` converts the `Value` floating point value to a `Currency` value. It checks whether `Value` is in the valid range of currencies (determined by `MinCurrency` ([1621](#)) and `MaxCurrency` ([1621](#))). If not, an `EConvertError` ([1812](#)) exception is raised.

Errors: If `Value` is out of range, an `EConvertError` ([1812](#)) exception is raised.

See also: `EConvertError` ([1812](#)), `TryFloatToCurr` ([1786](#)), `MinCurrency` ([1621](#)), `MaxCurrency` ([1621](#))

76.15.130 FloatToDateTime

Synopsis: Convert a float to a `TDateTime` value.

Declaration: `function FloatToDateTime (const Value: Extended) : TDateTime`

Visibility: default

Description: `FloatToDateTime` converts the `Value` floating point value to a `TDateTime` value. It checks whether `Value` is in the valid range of dates (determined by `MinDateTime` (1621) and `MaxDateTime` (1621)). If not, an `EConvertError` (1812) exception is raised.

Errors: If `Value` is out of range, an `EConvertError` (1812) exception is raised.

See also: `EConvertError` (1812), `MinDateTime` (1621), `MaxDateTime` (1621)

76.15.131 FloatToDecimal

Synopsis: Convert a float value to a `TFloatRec` value.

Declaration:

```
procedure FloatToDecimal(out Result: TFloatRec; const Value;
                        ValueType: TFloatValue; Precision: Integer;
                        Decimals: Integer)
procedure FloatToDecimal(out Result: TFloatRec; Value: Extended;
                        Precision: Integer; Decimals: Integer)
```

Visibility: default

Description: `FloatToDecimal` converts the float `Value` to a float description in the `ResultTFloatRec` (1632) format. It will store `Precision` digits in the `Digits` field, of which at most `Decimal` decimals.

Errors: None.

See also: `TFloatRec` (1632)

76.15.132 FloatToStr

Synopsis: Convert a float value to a string using a fixed format.

Declaration:

```
function FloatToStr(Value: Extended) : string
function FloatToStr(Value: Extended;
                    const FormatSettings: TFormatSettings) : string
function FloatToStr(Value: Double) : string
function FloatToStr(Value: Double;
                    const FormatSettings: TFormatSettings) : string
function FloatToStr(Value: Single) : string
function FloatToStr(Value: Single;
                    const FormatSettings: TFormatSettings) : string
function FloatToStr(Value: Currency) : string
function FloatToStr(Value: Currency;
                    const FormatSettings: TFormatSettings) : string
function FloatToStr(Value: Comp) : string
function FloatToStr(Value: Comp; const FormatSettings: TFormatSettings)
                    : string
function FloatToStr(Value: Int64) : string
function FloatToStr(Value: Int64; const FormatSettings: TFormatSettings)
                    : string
```

Visibility: default

Description: `FloatToStr` converts the floating point variable `Value` to a string representation. It will choose the shortest possible notation of the two following formats:

Fixed format will represent the string in fixed notation,

Decimal format will represent the string in scientific notation.

More information on these formats can be found in `FloatToStrF` (1710). `FloatToStr` is completely equivalent to the following call:

```
FloatToStrF(Value, ffGeneral, 15, 0);
```

Note that on unix systems, the localization support must be enabled explicitly, see `Localization` (1610).

Errors: None.

See also: `FloatToStrF` (1710), `FormatFloat` (1724), `StrToFloat` (1772)

Listing: ./examples/sysutex/ex67.pp

Program Example67;

{ This program demonstrates the FloatToStr function }

Uses sysutils;

Procedure Testit (Value : Extended);

begin

Writeln (Value, ' -> ', **FloatToStr**(Value));

Writeln (-Value, ' -> ', **FloatToStr**(-Value));

end;

Begin

 Testit (0.0);

 Testit (1.1);

 Testit (1.1e-3);

 Testit (1.1e-20);

 Testit (1.1e-200);

 Testit (1.1e+3);

 Testit (1.1e+20);

 Testit (1.1e+200);

End.

76.15.133 FloatToStrF

Synopsis: Convert a float value to a string using a given format.

Declaration:

```
function FloatToStrF(Value: Extended; format: TFloatFormat;
    Precision: Integer; Digits: Integer) : string
function FloatToStrF(Value: Extended; format: TFloatFormat;
    Precision: Integer; Digits: Integer;
    const FormatSettings: TFormatSettings) : string
function FloatToStrF(Value: Double; format: TFloatFormat;
    Precision: Integer; Digits: Integer) : string
function FloatToStrF(Value: Double; format: TFloatFormat;
    Precision: Integer; Digits: Integer;
    const FormatSettings: TFormatSettings) : string
function FloatToStrF(Value: Single; format: TFloatFormat;
    Precision: Integer; Digits: Integer) : string
```

```

function FloatToStrF(Value: Single; format: TFloatFormat;
    Precision: Integer; Digits: Integer;
    const FormatSettings: TFormatSettings) : string
function FloatToStrF(Value: Comp; format: TFloatFormat;
    Precision: Integer; Digits: Integer) : string
function FloatToStrF(Value: Comp; format: TFloatFormat;
    Precision: Integer; Digits: Integer;
    const FormatSettings: TFormatSettings) : string
function FloatToStrF(Value: Currency; format: TFloatFormat;
    Precision: Integer; Digits: Integer) : string
function FloatToStrF(Value: Currency; format: TFloatFormat;
    Precision: Integer; Digits: Integer;
    const FormatSettings: TFormatSettings) : string
function FloatToStrF(Value: Int64; format: TFloatFormat;
    Precision: Integer; Digits: Integer) : string
function FloatToStrF(Value: Int64; format: TFloatFormat;
    Precision: Integer; Digits: Integer;
    const FormatSettings: TFormatSettings) : string

```

Visibility: default

Description: `FloatToStrF` converts the floating point number value to a string representation, according to the settings of the parameters `Format`, `Precision` and `Digits`.

The meaning of the `Precision` and `Digits` parameter depends on the `Format` parameter. The format is controlled mainly by the `Format` parameter. It can have one of the following values:

ffCurrency Money format. Value is converted to a string using the global variables `CurrencyString`, `CurrencyFormat` and `NegCurrFormat`. The `Digits` parameter specifies the number of digits following the decimal point and should be in the range -1 to 18. If `Digits` equals -1, `CurrencyDecimals` is assumed. The `Precision` parameter is ignored.

ffExponent Scientific format. Value is converted to a string using scientific notation: 1 digit before the decimal point, possibly preceded by a minus sign if `Value` is negative. The number of digits after the decimal point is controlled by `Precision` and must lie in the range 0 to 15.

ffFixed Fixed point format. Value is converted to a string using fixed point notation. The result is composed of all digits of the integer part of `Value`, preceded by a minus sign if `Value` is negative. if `Digits` is positive, then following the integer part is `DecimalSeparator` and then the fractional part of `Value`, rounded off to `Digits` numbers. If the number is too large then the result will be in scientific notation.

ffGeneral General number format. The argument is converted to a string using `ffExponent` or `ffFixed` format, depending on which one gives the shortest string. There will be no trailing zeroes. If `Value` is less than 0.00001 or if the number of decimals left of the decimal point is larger than `Precision` then scientific notation is used, and `Digits` is the minimum number of digits in the exponent. Otherwise `Digits` is ignored. The minimum precision is 2.

ffNumber Is the same as `ffFixed`, except that thousand separators are inserted in the resulting string.

Errors: None.

See also: `FloatToStr` ([1709](#)), `FloatToText` ([1713](#))

Listing: `./examples/sysutex/ex68.pp`

Program `Example68`;

{ This program demonstrates the FloatToStrF function }

Uses sysutils;

Const Fmt : **Array** [TFloatFormat] **of** **string**[10] =
('general','exponent','fixed','number','Currency');

Procedure Testit (Value : Extended);

Var I,J : longint;
FF : TFloatFormat;

begin

For I:=5 **to** 15 **do**

For J:=1 **to** 4 **do**

For FF:=ffGeneral **to** {ffCurrency} ffGeneral **do**

begin

Write (Value,(Prec: ',I:2,', Dig: ',J,', fmt: ',Fmt[ff],') : ');

Writeln (FloatToStrF(Value,FF,I,J));

Write (-Value,(Prec: ',I:2,', Dig: ',J,', fmt: ',Fmt[ff],') : ');

Writeln (FloatToStrF(-Value,FF,I,J));

end;

end;

procedure Test2;

var

 F : Double;

begin

 F:=0.99411;

Writeln('0,0: ',floattostrF(F,FFgeneral,0,0));//=> 0.99,

Writeln('1,0: ',floattostrF(F,FFgeneral,1,0));//=> 0.99

Writeln('2,0: ',floattostrF(F,FFgeneral,2,0));//=> 0.99

Writeln('3,0: ',floattostrF(F,FFgeneral,3,0));//=> 0.994

 F:=0.99511;

Writeln('0,0: ',floattostrF(F,FFgeneral,0,0));//=> 1

Writeln('1,0: ',floattostrF(F,FFgeneral,1,0));//=> 1

Writeln('2,0: ',floattostrF(F,FFgeneral,2,0));//=> 1

Writeln('3,0: ',floattostrF(F,FFgeneral,3,0));//=> 0.995

 F:=12.145;

Writeln('0,0: ',floattostrF(F,FFgeneral,0,0));//=> 12

Writeln('1,0: ',floattostrF(F,FFgeneral,1,0));//=> 12

Writeln('2,0: ',floattostrF(F,FFgeneral,2,0));//=> 12

Writeln('3,0: ',floattostrF(F,FFgeneral,3,0));//=> 12.1

Writeln('4,0: ',floattostrF(F,FFgeneral,4,0));//=> 12.15

Writeln('5,0: ',floattostrF(F,FFgeneral,5,0));//=> 12.145

end;

Begin

 Testit (1.1);

 Testit (1.1E1);

 Testit (1.1E-1);

 Testit (1.1E5);

 Testit (1.1E-5);

 Testit (1.1E10);

 Testit (1.1E-10);

 Testit (1.1E15);

```

Testit (1.1E-15));
Testit (1.1E100));
Testit (1.1E-100));
End.

```

76.15.134 FloatToText

Synopsis: Return a string representation of a float, with a given format.

Declaration:

```

function FloatToText (Buffer: PWideChar; Value: Extended;
                     format: TFloatFormat; Precision: Integer;
                     Digits: Integer) : LongInt
function FloatToText (Buffer: PWideChar; Value: Extended;
                     format: TFloatFormat; Precision: Integer;
                     Digits: Integer;
                     const FormatSettings: TFormatSettings) : LongInt
function FloatToText (Buffer: PAnsiChar; Value: Extended;
                     format: TFloatFormat; Precision: Integer;
                     Digits: Integer) : LongInt
function FloatToText (Buffer: PAnsiChar; Value: Extended;
                     format: TFloatFormat; Precision: Integer;
                     Digits: Integer;
                     const FormatSettings: TFormatSettings) : LongInt

```

Visibility: default

Description: `FloatToText` converts the floating point variable `Value` to a string representation and stores it in `Buffer`. The conversion is governed by `format`, `Precision` and `Digits`. more information on these parameters can be found in `FloatToStrF` (1710). `Buffer` should point to enough space to hold the result. No checking on this is performed.

The result is the number of characters that was copied in `Buffer`.

Errors: None.

See also: `FloatToStr` (1709), `FloatToStrF` (1710)

Listing: ./examples/sysutex/ex69.pp

Program Example68;

{ This program demonstrates the FloatToStrF function }

Uses sysutils;

Const Fmt : **Array** [TFloatFormat] **of** string[10] =
 ('general','exponent','fixed','number','Currency');

Procedure Testit (Value : Extended);

Var I,J : longint;
 FF : TFloatFormat;
 S : ShortString;

begin
 For I:=5 **to** 15 **do**
 For J:=1 **to** 4 **do**

```

For FF:=ffgeneral to ffcurrency do
  begin
    Write (Value,'(Prec: ',I:2,', Dig: ',J,', fmt : ',Fmt[ff],') : ');
    SetLength(S,FloatToText (@S[1],Value,FF,I,J));
    Writeln (S);
    Write (-Value,'(Prec: ',I:2,', Dig: ',J,', fmt : ',Fmt[ff],') : ');
    SetLength(S,FloatToText (@S[1],-Value,FF,I,J));
    Writeln (S);
  end;
end;

Begin
  Testit (1.1);
  Testit (1.1E1);
  Testit (1.1E-1);
  Testit (1.1E5);
  Testit (1.1E-5);
  Testit (1.1E10);
  Testit (1.1E-10);
  Testit (1.1E15);
  Testit (1.1E-15);
  Testit (1.1E100);
  Testit (1.1E-100);
End.

```

76.15.135 FloatToTextFmt

Synopsis: Convert a float value to a string using a given mask.

Declaration: `function FloatToTextFmt(Buffer: PChar; Value: Extended; format: PChar; const FormatSettings: TFormatSettings) : Integer`
`function FloatToTextFmt(Buffer: PChar; Value: Extended; format: PChar) : Integer`

Visibility: default

Description: `FloatToTextFmt` returns a textual representation of `Value` in the memory location pointed to by `Buffer`. it uses the formatting specification in `Format` to do this. The return value is the number of characters that were written in the buffer.

For a list of valid formatting characters, see `FormatFloat` ([1724](#))

Errors: No length checking is performed on the buffer. The buffer should point to enough memory to hold the complete string. If this is not the case, an access violation may occur.

See also: `FormatFloat` ([1724](#))

76.15.136 FmtStr

Synopsis: Format a string with given arguments.

Declaration: `procedure FmtStr(var Res: string; const Fmt: string; const args: Array of const)`
`procedure FmtStr(var Res: string; const Fmt: string; const args: Array of const; const FormatSettings: TFormatSettings)`

Visibility: default

Description: `FmtStr` calls `Format` (1715) with `Fmt` and `Args` as arguments, and stores the result in `Res`. For more information on how the resulting string is composed, see `Format` (1715).

Errors: In case of error, a `EConvertError` (1812) exception is raised.

See also: `Format` (1715), `FormatBuf` (1722), `EConvertError` (1812)

Listing: `./examples/sysutex/ex70.pp`

Program Example70;

{ This program demonstrates the FmtStr function }

Uses sysutils;

Var S : AnsiString;

Begin

S:='';

FmtStr (S,'For some nice examples of fomattting see %s.',['Format']);

Writeln (S);

End.

76.15.137 ForceDirectories

Synopsis: Create a chain of directories.

Declaration: `function ForceDirectories(const Dir: RawByteString) : Boolean`
`function ForceDirectories(const Dir: UnicodeString) : Boolean`

Visibility: default

Description: `ForceDirectories` tries to create any missing directories in `Dir` till the whole path in `Dir` exists. It returns `True` if `Dir` already existed or was created successfully. If it failed to create any of the parts, `False` is returned.

76.15.138 Format

Synopsis: Format a string with given arguments.

Declaration: `function Format(const Fmt: AnsiString; const Args: Array of const)`
`: AnsiString`
`function Format(const Fmt: Ansistring; const Args: Array of const;`
`const FormatSettings: TFormatSettings) : Ansistring`

Visibility: default

Description: `Format` replaces all placeholders in `Fmt` with the arguments passed in `Args` and returns the resulting string. A placeholder looks as follows:

`'%' [[Index] ':' ] ['-'] [Width] ['.' Precision] ArgType`

elements between single quotes must be typed as shown without the quotes, and elements between square brackets `[ ]` are optional. The meaning of the different elements are shown below:

'%' starts the placeholder. If you want to insert a literal % character, then you must insert two of them : %%.

Index ':' takes the `Index`-th element in the argument array as the element to insert. If `index` is omitted, then the zeroth argument is taken.

'-' tells `Format` to left-align the inserted text. The default behaviour is to right-align inserted text. This can only take effect if the `Width` element is also specified.

Width the inserted string must have at least `Width` characters. If not, the inserted string will be padded with spaces. By default, the string is left-padded, resulting in a right-aligned string. This behaviour can be changed by the usage of the ' - ' character.

'.' Precision Indicates the precision to be used when converting the argument. The exact meaning of this parameter depends on `ArgType`.

The `Index`, `Width` and `Precision` parameters can be replaced by `*`, in which case their value will be read from the next element in the `Args` array. This value must be an integer, or an `EConvert-Error` (1812) exception will be raised.

The argument type is determined from `ArgType`. It can have one of the following values (case insensitive):

B Boolean format. The next argument in the `Args` array should be a boolean or an integer (a nonzero value is taken as `True`, zero is taken as false). The argument is converted to a string using `BoolToStr` (1664) with `True` as the second argument, i.e. boolean strings are used. If a precision is specified, then the string will have at most `Precision` characters in it, as for the "S" format specifier.

D Decimal format. The next argument in the `Args` array should be an integer. The argument is converted to a decimal string. If precision is specified, then the string will have at least `Precision` digits in it. If needed, the string is (left) padded with zeroes.

E Scientific format. The next argument in the `Args` array should be a Floating point value. The argument is converted to a decimal string using scientific notation, using `FloatToStrF` (1710), where the optional precision is used to specify the total number of decimals. (default a value of 15 is used). The exponent is formatted using maximally 3 digits.

In short, the E specifier formats it's argument as follows:

```
FloatToStrF (Argument, ffExponent, Precision, 3)
```

F Fixed point format. The next argument in the `Args` array should be a floating point value. The argument is converted to a decimal string, using fixed notation (see `FloatToStrF` (1710)). `Precision` indicates the number of digits following the decimal point.

In short, the F specifier formats its argument as follows:

```
FloatToStrF (Argument, ffFixed, ffixed, 9999, Precision)
```

G General number format. The next argument in the `Args` array should be a floating point value. The argument is converted to a decimal string using fixed point notation or scientific notation, depending on which gives the shortest result. `Precision` is used to determine the number of digits after the decimal point.

In short, the G specifier formats it's argument as follows:

```
FloatToStrF (Argument, ffGeneral, Precision, 3)
```

M Currency format. the next argument in the `Args` array must be a floating point value. The argument is converted to a decimal string using currency notation. This means that fixed-point notation is used, but that the currency symbol is appended. If precision is specified, then then it overrides the `CurrencyDecimals` global variable used in the `FloatToStrF` (1710)

In short, the M specifier formats it's argument as follows:

`FloatToStrF (Argument, ffCurrency, 9999, Precision)`

N Number format. This is the same as fixed point format, except that thousand separators are inserted in the resulting string.

P Pointer format. The next argument in the `Args` array must be a pointer (typed or untyped). The pointer value is converted to a string of length 8, representing the hexadecimal value of the pointer.

S String format. The next argument in the `Args` array must be a string or a character (unicode or ansi is acceptable for both). The argument is simply copied to the result string. If `Precision` is specified, then only `Precision` characters are copied to the result string.

U Unsigned decimal format. The next argument in the `Args` array should be an unsigned integer. The argument is converted to a decimal string. If precision is specified, then the string will have at least `Precision` digits in it. If needed, the string is (left) padded with zeroes.

X hexadecimal format. The next argument in the `Args` array must be an integer. The argument is converted to a hexadecimal string with just enough characters to contain the value of the integer. If `Precision` is specified then the resulting hexadecimal representation will have at least `Precision` characters in it (with a maximum value of 32).

An unknown formatting character is ignored.

Errors: In case of error, an `EConvertError` exception is raised. Possible errors are:

- 1.Errors in the format specifiers.
- 2.The next argument is not of the type needed by a specifier.
- 3.The number of arguments is not sufficient for all format specifiers.

See also: `FormatBuf` ([1722](#)), `EConvertError` ([1812](#))

Listing: `./examples/sysutex/ex71.pp`

Program `example71;`

`{ $mode objfpc }`

`{ This program demonstrates the Format function }`

Uses `sysutils;`

Var `P : Pointer;`
`fmt,S : string;`

`{ Expected output:`

```

    [%d] => [10]
    [%%] => [%]
    [%10d] => [ 10]
    [%.4d] => [0010]
    [%10.4d] => [ 0010]
    [%0:d] => [10]
    [%0:10d] => [ 10]
    [%0:10.4d] => [ 0010]
    [%0:-10d] => [10 ]
    [%0:-10.4d] => [0010 ]
    [%-*.*d] => [00010]

```

`}`

Procedure `TestInteger;`
begin

Try

```

Fmt:='%d';S:=Format (Fmt,[10]);writeln(Fmt:12,' => ',s);
Fmt:='%%';S:=Format (Fmt,[10]);writeln(Fmt:12,' => ',s);
Fmt:='%10d';S:=Format (Fmt,[10]);writeln(Fmt:12,' => ',s);
fmt:='%4d';S:=Format (fmt,[10]);writeln(Fmt:12,' => ',s);
Fmt:='%10.4d';S:=Format (Fmt,[10]);writeln(Fmt:12,' => ',s);
Fmt:='%0:d';S:=Format (Fmt,[10]);writeln(Fmt:12,' => ',s);
Fmt:='%0:10d';S:=Format (Fmt,[10]);writeln(Fmt:12,' => ',s);
Fmt:='%0:10.4d';S:=Format (Fmt,[10]);writeln(Fmt:12,' => ',s);
Fmt:='%0:-10d';S:=Format (Fmt,[10]);writeln(Fmt:12,' => ',s);
Fmt:='%0:-10.4d';S:=Format (fmt,[10]);writeln(Fmt:12,' => ',s);
Fmt:='%-*.d';S:=Format (fmt,[4,5,10]);writeln(Fmt:12,' => ',s);

```

except

```

On E : Exception do

```

```

  begin

```

```

    Writeln ('Exception caught : ',E.Message);

```

```

  end;

```

```

end;

```

```

writeln ('Press enter');

```

```

readln;

```

```

end;

```

```

{ Expected output:

```

```

    [%x] => [A]
    [%10x] => [ A]
    [%10.4x] => [ 000A]
    [%0:x] => [A]
    [%0:10x] => [ A]
    [%0:10.4x] => [ 000A]
    [%0:-10x] => [ A ]
    [%0:-10.4x] => [000A ]
    [%-*.x] => [0000A]
}

```

```

}

```

```

Procedure TestHexaDecimal;

```

```

begin

```

```

  try

```

```

    Fmt:='%x';S:=Format (Fmt,[10]);writeln(Fmt:12,' => ',s);
    Fmt:='%10x';S:=Format (Fmt,[10]);writeln(Fmt:12,' => ',s);
    Fmt:='%10.4x';S:=Format (Fmt,[10]);writeln(Fmt:12,' => ',s);
    Fmt:='%0:x';S:=Format (Fmt,[10]);writeln(Fmt:12,' => ',s);
    Fmt:='%0:10x';S:=Format (Fmt,[10]);writeln(Fmt:12,' => ',s);
    Fmt:='%0:10.4x';S:=Format (Fmt,[10]);writeln(Fmt:12,' => ',s);
    Fmt:='%0:-10x';S:=Format (Fmt,[10]);writeln(Fmt:12,' => ',s);
    Fmt:='%0:-10.4x';S:=Format (fmt,[10]);writeln(Fmt:12,' => ',s);
    Fmt:='%-*.x';S:=Format (fmt,[4,5,10]);writeln(Fmt:12,' => ',s);

```

except

```

On E : Exception do

```

```

  begin

```

```

    Writeln ('Exception caught : ',E.Message);

```

```

  end;

```

```

end;

```

```

writeln ('Press enter');

```

```

readln;

```

```

end;

```

```

{ Expected output:

```

```

    [0x%p] => [0x0012D687]
    [0x%10p] => [0x 0012D687]

```

```

[0x%10.4p] => [0x 0012D687]
[0x%0:p] => [0x0012D687]
[0x%0:10p] => [0x 0012D687]
[0x%0:10.4p] => [0x 0012D687]
[0x%0:-10p] => [0x0012D687]
[0x%0:-10.4p] => [0x0012D687]
[%-*. *p] => [0012D687]
}
Procedure TestPointer;
begin
  P:=Pointer(1234567);
  try
    Fmt:='[0x%p]';S:=Format (Fmt,[P]);writeln(Fmt:12,' => ',s);
    Fmt:='[0x%10p]';S:=Format (Fmt,[P]);writeln(Fmt:12,' => ',s);
    Fmt:='[0x%10.4p]';S:=Format (Fmt,[P]);writeln(Fmt:12,' => ',s);
    Fmt:='[0x%0:p]';S:=Format (Fmt,[P]);writeln(Fmt:12,' => ',s);
    Fmt:='[0x%0:10p]';S:=Format (Fmt,[P]);writeln(Fmt:12,' => ',s);
    Fmt:='[0x%0:10.4p]';S:=Format (Fmt,[P]);writeln(Fmt:12,' => ',s);
    Fmt:='[0x%0:-10p]';S:=Format (Fmt,[P]);writeln(Fmt:12,' => ',s);
    Fmt:='[0x%0:-10.4p]';S:=Format (Fmt,[P]);writeln(Fmt:12,' => ',s);
    Fmt:='[%-*. *p]';S:=Format (Fmt,[4,5,P]);writeln(Fmt:12,' => ',s);
  except
    On E : Exception do
      begin
        Writeln ('Exception caught : ',E.Message);
      end;
  end;
  writeln ('Press enter');
  readln;
end;

{ Expected output:
  [%s]=> [This is a string]
  [%0:s]=> [This is a string]
  [%0:18s]=> [ This is a string]
  [%0:-18s]=> [This is a string ]
  [%0:18.12s]=> [ This is a st]
  [%-*. *s]=> [This is a st ]
}

Procedure TestString;
begin
  try
    Fmt:='[%s]';S:=Format(fmt,['This is a string']);Writeln(fmt:12,'=> ',s);
    fmt:='[%0:s]';s:=Format(fmt,['This is a string']);Writeln(fmt:12,'=> ',s);
    fmt:='[%0:18s]';s:=Format(fmt,['This is a string']);Writeln(fmt:12,'=> ',s);
    fmt:='[%0:-18s]';s:=Format(fmt,['This is a string']);Writeln(fmt:12,'=> ',s);
    fmt:='[%0:18.12s]';s:=Format(fmt,['This is a string']);Writeln(fmt:12,'=> ',s);
    fmt:='[%-*. *s]';s:=Format(fmt,[18,12,'This is a string']);Writeln(fmt:12,'=> ',s);
  except
    On E : Exception do
      begin
        Writeln ('Exception caught : ',E.Message);
      end;
  end;
  writeln ('Press enter');
  readln;
end;

```


{ Expected output:

```

    [%e] => [1.2340000000000000E+000]
    [%10e] => [1.2340000000000000E+000]
    [%10.4e] => [1.234E+000]
    [%0:e] => [1.2340000000000000E+000]
    [%0:10e] => [1.2340000000000000E+000]
    [%0:10.4e] => [1.234E+000]
    [%0:-10e] => [1.2340000000000000E+000]
    [%0:-10.4e] => [1.234E+000]
    [%-*. *e] => [1.2340E+000]

```

}

Procedure TestExponential;

begin

Try

```

    Fmt:='%e';S:=Format (Fmt,[1.234]);writeln(Fmt:12,' => ',s);
    Fmt:='%10e';S:=Format (Fmt,[1.234]);writeln(Fmt:12,' => ',s);
    Fmt:='%10.4e';S:=Format (Fmt,[1.234]);writeln(Fmt:12,' => ',s);
    Fmt:='%0:e';S:=Format (Fmt,[1.234]);writeln(Fmt:12,' => ',s);
    Fmt:='%0:10e';S:=Format (Fmt,[1.234]);writeln(Fmt:12,' => ',s);
    Fmt:='%0:10.4e';S:=Format (Fmt,[1.234]);writeln(Fmt:12,' => ',s);
    Fmt:='%0:-10e';S:=Format (Fmt,[1.234]);writeln(Fmt:12,' => ',s);
    Fmt:='%0:-10.4e';S:=Format (fmt,[1.234]);writeln(Fmt:12,' => ',s);
    Fmt:='%-*. *e';S:=Format (fmt,[4,5,1.234]);writeln(Fmt:12,' => ',s);

```

except

On E : Exception **do**

begin

Writeln ('Exception caught : ',E.Message);

end;

end;

writeln ('Press enter');

readln;

end;

{ Expected output:

```

    [%e] => [-1.2340000000000000E+000]
    [%10e] => [-1.2340000000000000E+000]
    [%10.4e] => [-1.234E+000]
    [%0:e] => [-1.2340000000000000E+000]
    [%0:10e] => [-1.2340000000000000E+000]
    [%0:10.4e] => [-1.234E+000]
    [%0:-10e] => [-1.2340000000000000E+000]
    [%0:-10.4e] => [-1.234E+000]
    [%-*. *e] => [-1.2340E+000]

```

}

Procedure TestNegativeExponential;

begin

Try

```

    Fmt:='%e';S:=Format (Fmt,[-1.234]);writeln(Fmt:12,' => ',s);
    Fmt:='%10e';S:=Format (Fmt,[-1.234]);writeln(Fmt:12,' => ',s);
    Fmt:='%10.4e';S:=Format (Fmt,[-1.234]);writeln(Fmt:12,' => ',s);
    Fmt:='%0:e';S:=Format (Fmt,[-1.234]);writeln(Fmt:12,' => ',s);
    Fmt:='%0:10e';S:=Format (Fmt,[-1.234]);writeln(Fmt:12,' => ',s);
    Fmt:='%0:10.4e';S:=Format (Fmt,[-1.234]);writeln(Fmt:12,' => ',s);
    Fmt:='%0:-10e';S:=Format (Fmt,[-1.234]);writeln(Fmt:12,' => ',s);
    Fmt:='%0:-10.4e';S:=Format (fmt,[-1.234]);writeln(Fmt:12,' => ',s);
    Fmt:='%-*. *e';S:=Format (fmt,[4,5,-1.234]);writeln(Fmt:12,' => ',s);

```

except

On E : Exception **do**

```

    begin
    Writeln ('Exception caught : ',E.Message);
    end;
end;
writeln ('Press enter');
readln;
end;

{ Expected output:
    [%e] => [1.2340000000000000E-002]
    [%10e] => [1.2340000000000000E-002]
    [%10.4e] => [1.234E-002]
    [%0:e] => [1.2340000000000000E-002]
    [%0:10e] => [1.2340000000000000E-002]
    [%0:10.4e] => [1.234E-002]
    [%0:-10e] => [1.2300000000000000E-002]
    [%0:-10.4e] => [1.234E-002]
    [%-*.*e] => [1.2340E-002]
}
Procedure TestSmallExponential;
begin
    Try
        Fmt:='%e';S:=Format (Fmt,[0.01234]);writeln(Fmt:12,' => ',s);
        Fmt:='%10e';S:=Format (Fmt,[0.01234]);writeln(Fmt:12,' => ',s);
        Fmt:='%10.4e';S:=Format (Fmt,[0.01234]);writeln(Fmt:12,' => ',s);
        Fmt:='%0:e';S:=Format (Fmt,[0.01234]);writeln(Fmt:12,' => ',s);
        Fmt:='%0:10e';S:=Format (Fmt,[0.01234]);writeln(Fmt:12,' => ',s);
        Fmt:='%0:10.4e';S:=Format (Fmt,[0.01234]);writeln(Fmt:12,' => ',s);
        Fmt:='%0:-10e';S:=Format (Fmt,[0.01234]);writeln(Fmt:12,' => ',s);
        Fmt:='%0:-10.4e';S:=Format (Fmt,[0.01234]);writeln(Fmt:12,' => ',s);
        Fmt:='%-*.*e';S:=Format (Fmt,[4,5,0.01234]);writeln(Fmt:12,' => ',s);
    except
        On E : Exception do
            begin
                Writeln ('Exception caught : ',E.Message);
            end;
        end;
    writeln ('Press enter');
    readln;
end;

{ Expected output:
    [%e] => [-1.2340000000000000E-002]
    [%10e] => [-1.2340000000000000E-002]
    [%10.4e] => [-1.234E-002]
    [%0:e] => [-1.2340000000000000E-002]
    [%0:10e] => [-1.2340000000000000E-002]
    [%0:10.4e] => [-1.234E-002]
    [%0:-10e] => [-1.2340000000000000E-002]
    [%0:-10.4e] => [-1.234E-002]
    [%-*.*e] => [-1.2340E-002]
}
Procedure TestSmallNegExponential;
begin
    Try
        Fmt:='%e';S:=Format (Fmt,[-0.01234]);writeln(Fmt:12,' => ',s);
        Fmt:='%10e';S:=Format (Fmt,[-0.01234]);writeln(Fmt:12,' => ',s);
        Fmt:='%10.4e';S:=Format (Fmt,[-0.01234]);writeln(Fmt:12,' => ',s);

```

```

Fmt:='%0:e';S:=Format (Fmt,[-0.01234]);writeln(Fmt:12,' => ',s);
Fmt:='%0:10e';S:=Format (Fmt,[-0.01234]);writeln(Fmt:12,' => ',s);
Fmt:='%0:10.4e';S:=Format (Fmt,[-0.01234]);writeln(Fmt:12,' => ',s);
Fmt:='%0:-10e';S:=Format (Fmt,[-0.01234]);writeln(Fmt:12,' => ',s);
Fmt:='%0:-10.4e';S:=Format (fmt,[-0.01234]);writeln(Fmt:12,' => ',s);
Fmt:='%-*.e';S:=Format (fmt,[4,5,-0.01234]);writeln(Fmt:12,' => ',s);
except
  On E : Exception do
    begin
      Writeln ('Exception caught : ',E.Message);
    end;
end;
writeln ('Press enter');
readln;
end;

begin
  TestInteger;
  TestHexadecimal;
  TestPointer;
  teststring;
  TestExponential;
  TestNegativeExponential;
  TestSmallExponential;
  TestSmallNegExponential;
end.

```

76.15.139 FormatBuf

Synopsis: Format a string with given arguments and store the result in a buffer.

Declaration: `function FormatBuf(var Buffer; BufLen: Cardinal; const Fmt;
 fmtLen: Cardinal; const Args: Array of const)
 : Cardinal`

`function FormatBuf(var Buffer; BufLen: Cardinal; const Fmt;
 fmtLen: Cardinal; const Args: Array of const;
 const FormatSettings: TFormatSettings) : Cardinal`

Visibility: default

Description: `FormatBuf` calls `Format` (1715) and stores the result in `Buf`.

See also: `Format` (1715)

Listing: `./examples/sysutex/ex72.pp`

Program Example72;

{ This program demonstrates the FormatBuf function }

Uses sysutils;

Var

S : ShortString;

Const

Fmt : ShortString = 'For some nice examples of fomatting see %s.';

```

Begin
  S:="";
  SetLength(S,FormatBuf (S[1],255,Fmt[1],Length(Fmt),['Format']));
  Writeln (S);
End.

```

76.15.140 FormatCurr

Synopsis: Format a currency.

Declaration: `function FormatCurr(const Format: string; Value: Currency) : string`
`function FormatCurr(const Format: string; Value: Currency;`
`const FormatSettings: TFormatSettings) : string`

Visibility: default

Description: `FormatCurr` formats the currency `Value` according to the formatting rule in the `Format` parameter, and returns the resulting string.

For an explanation of the formatting characters usable in the `Format` parameter, see `FormatFloat` (1724).

See also: `FormatFloat` (1724), `FloatToText` (1713)

76.15.141 FormatDateTime

Synopsis: Return a string representation of a `TDateTime` value with a given format.

Declaration: `function FormatDateTime(const FormatStr: string; DateTime: TDateTime;`
`Options: TFormatDateTimeOptions) : string`
`function FormatDateTime(const FormatStr: string; DateTime: TDateTime;`
`const FormatSettings: TFormatSettings;`
`Options: TFormatDateTimeOptions) : string`

Visibility: default

Description: `FormatDateTime` formats the date and time encoded in `DateTime` according to the formatting given in `FormatStr`. The complete list of formatting characters can be found in `formatchars` (1615).

When the format string is empty, 'c' is used instead.

Note that on unix systems, the localization support must be enabled explicitly, see `Localization` (1610).

Errors: On error (such as an invalid character in the formatting string), and `EConvertError` (1812) exception is raised.

See also: `DateTimeToStr` (1674), `DateToStr` (1677), `TimeToStr` (1783), `StrToDateTime` (1770), `EConvertError` (1812), `Localization` (1610)

Listing: `./examples/sysutex/ex14.pp`

Program Example14;

{ This program demonstrates the FormatDateTime function }

Uses sysutils;

Var ThisMoment : TDateTime;

Begin

 ThisMoment:=**Now**;

Writeln ('Now : ',**FormatDateTime**('hh:nn',ThisMoment));

Writeln ('Now : ',**FormatDateTime**('DD MM YYYY',ThisMoment));

Writeln ('Now : ',**FormatDateTime**('c',ThisMoment));

End.

76.15.142 FormatFloat

Synopsis: Format a float according to a certain mask.

Declaration: `function FormatFloat(const Format: string; Value: Extended) : string`
`function FormatFloat(const Format: string; Value: Extended;`
`const FormatSettings: TFormatSettings) : string`

Visibility: default

Description: `FormatFloat` formats the floating-point value given by `Value` using the format specifications in `Format`. The format specifier can give format specifications for positive, negative or zero values (separated by a semicolon). Note that the minus sign must be explicitly specified in the section for negative numbers:

```
Writeln(FormatFloat('#0.00;#0.00-;0",FF'));
```

If the format specifier is empty or the value needs more than 18 digits to be correctly represented, the result is formatted with a call to `FloatToStrF` (1710) with the `ffGeneral` format option.

The following format specifiers are supported:

0 is a digit place holder. If there is a corresponding digit in the value being formatted, then it replaces the 0. If not, the 0 is left as-is.

is also a digit place holder. If there is a corresponding digit in the value being formatted, then it replaces the #. If not, it is removed. by a space.

. determines the location of the decimal point. Only the first **'.'** character is taken into account. If the value contains digits after the decimal point, then it is replaced by the value of the `DecimalSeparator` character.

, determines the use of the thousand separator character in the output string. If the format string contains one or more **','** characters, then thousand separators will be used. The `ThousandSeparator` character is used.

E+ determines the use of scientific notation. If **'E+'** or **'E-'** (or their lowercase counterparts) are present then scientific notation is used. The number of digits in the output string is determined by the number of 0 characters after the **'E+'**

; This character separates sections for positive, negative, and zero numbers in the format string.

Errors: If an error occurs, an exception is raised.

See also: `FloatToStr` (1709)

Listing: ./examples/sysutex/ex89.pp

Program Example89;

{ This program demonstrates the FormatFloat function }

Uses sysutils;

Const

```
NrFormat=9;
FormatStrings : Array[1..NrFormat] of string = (
    ' ',
    '0',
    '0.00',
    '#.##',
    '##0.00',
    '##0.00;(##0.00)',
    '##0.00;;Zero',
    '0.000E+00',
    '###E-0');
NrValue = 5;
FormatValues : Array[1..NrValue] of Double =
    (1234,-1234,0.5,0,-0.5);

Width = 12;
FWidth = 20;
```

Var

```
I,J : Integer;
S : String;
```

begin

```
Write('Format':FWidth);
For I:=1 to NrValue do
    Write(FormatValues[i]:Width:2);
Writeln;
For I:=1 to NrFormat do
    begin
        Write(FormatStrings[i]:FWidth);
        For J:=1 to NrValue do
            begin
                S:=FormatFloat(FormatStrings[I],FormatValues[J]);
                Write(S:Width);
            end;
        Writeln;
    end;
```

End.

76.15.143 FreeAndNil

Synopsis: Free object if needed, and set object reference to Nil.

Declaration: procedure FreeAndNil (var obj)

Visibility: default

Description: FreeAndNil will free the object in Obj and will set the reference in Obj to Nil. The reference is set to Nil first, so if an exception occurs in the destructor of the object, the reference will be Nil anyway.

Errors: Exceptions that occur during the destruction of `Obj` are not caught.

76.15.144 **FreeMemAndNil**

Synopsis: Free the heap memory pointed to by a pointer and set pointer to nil.

Declaration: `procedure FreeMemAndNil (var p)`

Visibility: default

Description: `FreeMemAndNil` will free the memory pointed to by `P` and will set `P` to `Nil`. Note that this function is not type safe; the programmer is responsible for ensuring that this function is called with a correct pointer parameter. Failure to do so may result in run-time errors.

See also: `FreeAndNil` ([1725](#))

76.15.145 **GetAppConfigDir**

Synopsis: Return the appropriate directory for the application's configuration files.

Declaration: `function GetAppConfigDir (Global: Boolean) : string`

Visibility: default

Description: `GetAppConfigDir` returns the name of a directory in which the application should store its configuration files on the current OS. If the parameter `Global` is `True` then the directory returned is a global directory, i.e. valid for all users on the system. If the parameter `Global` is false, then the directory is specific for the user who is executing the program. On systems that do not support multi-user environments, these two directories may be the same.

The directory which is returned is the name of the directory where the application is supposed to store files. This does not mean that the directory exists, or that the user can write in this directory (especially if `Global=True`). It just returns the name of the appropriate location. Also note that the returned name always contains an ending path delimiter.

On systems where the operating system provides a call to determine this location, this call will be used. On systems where there is no such call, an algorithm is used which reflects common practice on that system.

The application name is deduced from the binary name via the `ApplicationName` ([1662](#)) call, but can be configured by means of the `OnGetApplicationName` ([1645](#)) callback.

If `VendorName` ([1796](#)) is not-empty, then `VendorName` will also be inserted before the application-specific directory.

Errors: None.

See also: `GetAppConfigFile` ([1726](#)), `ApplicationName` ([1662](#)), `OnGetApplicationName` ([1645](#)), `CreateDir` ([1671](#)), `SysConfigDir` ([1624](#)), `VendorName` ([1796](#))

76.15.146 **GetAppConfigFile**

Synopsis: Return an appropriate name for an application configuration file.

Declaration: `function GetAppConfigFile (Global: Boolean) : string`
`function GetAppConfigFile (Global: Boolean; SubDir: Boolean) : string`

Visibility: default

Description: `GetAppConfigFile` returns the name of a file in which the application can store its configuration parameters. The exact name and location of the file depends on the customs of the operating system.

The `Global` parameter determines whether it is a global configuration file (value `True`) or a personal configuration file (value `False`).

The parameter `SubDir`, in case it is set to `True`, will insert the name of a directory before the filename. This can be used in case the application needs to store other data than configuration data in an application-specific directory. Default behaviour is to set this to `False`.

Note that on Windows, even when `Subdir` is `False`, a subdirectory is created for the application configuration files, as per the windows specifications. Specifying `true` will create a subdirectory of the application settings subdirectory.

The default file extension of the returned file is: `.cfg`

No assumptions should be made about the existence or writeability of this file, or the directory where the file should reside. It is best to call `ForceDirectories` (1715) prior to opening a file with the resulting filename.

On systems where the operating system provides a call to determine the location of configuration files, this call will be used. On systems where there is no such call, an algorithm is used which reflects common practice on that system.

The application name is deduced from the binary name via the `ApplicationName` (1662) call, but can be configured by means of the `OnGetApplicationName` (1645) callback.

If `VendorName` (1796) is not-empty, then `VendorName` will be inserted in the path for the config file directory.

Errors: None.

See also: `GetAppConfigDir` (1726), `OnGetApplicationName` (1645), `ApplicationName` (1662), `CreateDir` (1671), `ConfigExtension` (1617), `SysConfigDir` (1624), `VendorName` (1796)

76.15.147 `GetCompiledArchitecture`

Declaration: `function GetCompiledArchitecture : TOSVersion.TArchitecture`

Visibility: default

76.15.148 `GetCompiledPlatform`

Declaration: `function GetCompiledPlatform : TOSVersion.TPlatform`

Visibility: default

76.15.149 `GetCurrentDir`

Synopsis: Return the current working directory of the application.

Declaration: `function GetCurrentDir : AnsiString`

Visibility: default

Description: `GetCurrentDir` returns the current working directory.

Errors: None.

See also: `SetCurrentDir` (1750), `DiskFree` (1681), `DiskSize` (1682)

Listing: ./examples/sysutex/ex28.pp

Program Example28;

{ This program demonstrates the GetCurrentDir function }

Uses sysutils;

Begin

Writeln ('Current Directory is : ',GetCurrentDir);
End.

76.15.150 GetDirs

Synopsis: Return a list of directory names from a path.

Declaration: function GetDirs(var DirName: UnicodeString;
 var Dirs: Array of PUnicodeChar) : LongInt
function GetDirs(var DirName: RawByteString;
 var Dirs: Array of PAnsiChar) : LongInt

Visibility: default

Description: GetDirs splits DirName in a null-byte separated list of directory names, Dirs is an array of PChars, pointing to these directory names. The function returns the number of directories found, or -1 if none were found. DirName must contain only OSDirSeparator as Directory separator chars.

Errors: None.

See also: ExtractRelativePath ([1692](#))

Listing: ./examples/sysutex/ex45.pp

Program Example45;

{ This program demonstrates the GetDirs function }
{ \$H+ }

Uses sysutils;

Var Dirs : **Array**[0..127] **of** pchar;
 I,Count : longint;
 Dir,NewDir : **String**;

Begin

 Dir:=GetCurrentDir;
 Writeln ('Dir : ',Dir);
 NewDir:="";
 count:=GetDirs(Dir,Dirs);
 For I:=0 **to** Count-1 **do**
 begin
 NewDir:=NewDir+'/' + **StrPas**(Dirs[I]);
 Writeln (NewDir);
 end;
End.

76.15.151 GetDriveIDFromLetter

Synopsis: Return the drive ID based on a drive letter.

```
Declaration: function GetDriveIDFromLetter(const ADrive: RawByteString) : Byte
function GetDriveIDFromLetter(const ADrive: UnicodeString) : Byte
```

Visibility: default

Description: `GetDriveIDFromLetter` returns the drive ID based on a drive letter. The drive ID can then be used in the `DiskSize` (1682) and `DiskFree` (1681) calls.

See also: [DiskSize \(1682\)](#), [DiskFree \(1681\)](#)

76.15.152 GetEnvironmentString

Synopsis: Return an environment variable by index.

Declaration: `function GetEnvironmentString(Index: Integer) : AnsiString`

Visibility: default

Description: `GetEnvironmentString` returns the Index-th environment variable. The index is 1 based, and is bounded from above by the result of `GetEnvironmentVariableCount` (1729).

For an example, `GetEnvironmentVariableCount` ([1729](#)).

Remark Note that on Windows, environment strings can start with an equal sign (=). This is a trick used to pass the current working directory to a newly created process. In this case, extracting the variable name as the characters before the first equal sign will result in an empty name.

Errors: If there is no environment, an empty string is returned.

See also: [GetEnvironmentVariable \(1729\)](#), [GetEnvironmentVariableCount \(1729\)](#)

76.15.153 GetEnvironmentVariable

Synopsis: Return the value of an environment variable.

```
Declaration: function GetEnvironmentVariable(const EnvVar: AnsiString) : AnsiString  
                ; Overload  
function GetEnvironmentVariable(const EnvVar: UnicodeString)  
                : UnicodeString; Overload
```

Visibility: default

Description: `GetEnvironmentVariable` returns the value of the `EnvVar` environment variable. If the specified variable does not exist or `EnvVar` is empty, an empty string is returned.

See also: [GetEnvironmentString \(1729\)](#), [GetEnvironmentVariableCount \(1729\)](#)

76.15.154 GetEnvironmentVariableCount

Synopsis: Return the number of variables in the environment.

Declaration: function GetEnvironmentVariableCount : Integer

Visibility: default

Description: `GetEnvironmentVariableCount` returns the number of variables in the environment. The number is 1 based, but the result may be zero if there are no environment variables.

Errors: If there is no environment, -1 may be returned.

See also: `GetEnvironmentString` (1729), `GetEnvironmentVariable` (1729)

Listing: `./examples/sysutex/ex92.pp`

```
{ $h+ }
program example92;

{ This program demonstrates the
  GetEnvironmentVariableCount function }

uses sysutils;

Var
  I : Integer;

begin
  For I:=1 to GetEnvironmentVariableCount do
    Writeln(i:3, ' : ', GetEnvironmentString(i));
end.
```

76.15.155 GetFileAsString

Synopsis: Return the contents of a file as a string.

Declaration:

```
function GetFileAsString(const aFileName: RawByteString) : RawByteString
function GetFileAsString(const aFileName: RawByteString;
                          aEncoding: TEncoding) : RawByteString
function GetFileAsString(const aFileName: UnicodeString) : UnicodeString
function GetFileAsString(const aFileName: UnicodeString;
                          aEncoding: TEncoding) : UnicodeString
```

Visibility: default

Description: `GetFileAsString` returns the contents of `aFileName` or `aHandle` as a string. You can specify the string encoding in the `aEncoding` argument. If no encoding is specified `TEncoding.SystemEncoding` (1877) is used. The whole file is read in the case of `aHandle`: The file position is first set to offset zero.

Errors: If an invalid file handle or a file handle that does not support the seek operation, a `EInOutError` (1815) exception is raised.

See also: `GetFileContents` (1730), `TEncoding.SystemEncoding` (1877), `TEncoding.GetAnsiString` (1871)

76.15.156 GetFileContents

Synopsis: Return the contents of a file as an array of bytes.

Declaration:

```
function GetFileContents(const aFileName: RawByteString) : TBytes
function GetFileContents(const aFileName: UnicodeString) : TBytes
function GetFileContents(const aHandle: THandle) : TBytes
```

Visibility: default

Description: `GetFileContents` returns the contents of `aFileName` or `aHandle` as an array of bytes. The whole file is read in the case of `aHandle`; The file position is first set to offset zero.

Errors: If an invalid file handle or a file handle that does not support the seek operation, a `EInOutError` (1815) exception is raised.

See also: `GetFileAsString` (1730)

76.15.157 `GetFileHandle`

Synopsis: Extract OS handle from an untyped file or text file.

Declaration: `function GetFileHandle(var f: File) : THandle`
`function GetFileHandle(var f: Text) : THandle`

Visibility: default

Description: `GetFileHandle` returns the operating system handle for the file descriptor `F`. It can be used in various file operations which are not directly supported by the pascal language.

76.15.158 `GetLastError`

Synopsis: Return the last code from the OS.

Declaration: `function GetLastError : Integer`

Visibility: default

Description: `GetLastError` returns the error code from the last operating system call. It does not reset this code. In general, it should be called when an operating system call reported an error condition. In that case, `GetLastError` gives extended information about the error.

No assumptions should be made about the resetting of the error code by subsequent OS calls. This may be platform dependent.

See also: `RaiseLastError` (1747)

76.15.159 `GetLocalTime`

Synopsis: Get the local time.

Declaration: `procedure GetLocalTime(var SystemTime: TSystemTime)`

Visibility: default

Description: `GetLocalTime` returns the system time in a `TSystemTime` (1640) format.

Errors: None.

See also: `Now` (1745), `Date` (1673), `Time` (1781), `TSystemTime` (1640)

76.15.160 GetLocalTimeOffset

Synopsis: Return local timezone offset.

Declaration: `function GetLocalTimeOffset : Integer`
`function GetLocalTimeOffset(const DateTime: TDateTime;`
`const InputIsUTC: Boolean;`
`out Offset: Integer) : Boolean`
`function GetLocalTimeOffset(const DateTime: TDateTime;`
`const InputIsUTC: Boolean;`
`out Offset: Integer; out IsDST: Boolean)`
`: Boolean`
`function GetLocalTimeOffset(const DateTime: TDateTime;`
`const InputIsUTC: Boolean) : Integer`

Visibility: default

Description: `GetLocalTimeOffset` returns the local timezone offset in minutes. This is the difference between UTC time and local time:

$$\text{UTC} = \text{LocalTime} + \text{GetLocalTimeOffset}$$

The `TDateTime`-overloads return the offset at the specified `DateTime`. In case the `TDateTime`-aware function is not supported on the current platform, `False` is returned or it falls back to the offset to current time (depending on the overload used).

Note that on Linux/Unix, the information returned from the `TDateTime`-unaware function may be inaccurate around the DST time changes (for optimization). In that case, the `unix.ReReadLocalTime` (2220) unit must be used to re-initialize the timezone information.

The `TDateTime`-aware overloads are supported currently only on Windows Vista and newer.

See also: `unix.ReReadLocalTime` (2220), `Date` (1673), `Time` (1781), `Now` (1745)

76.15.161 GetModuleName

Synopsis: Return the name of the current module.

Declaration: `function GetModuleName(Module: HMODULE) : string`

Visibility: default

Description: `GetModuleName` returns the name of the current module. On windows, this is the name of the executable when executed in an executable, or the name of the library when executed in a library.

On all other platforms, the result is always empty, since they provide no such functionality.

76.15.162 GetTempDir

Synopsis: Return name of system's temporary directory.

Declaration: `function GetTempDir(Global: Boolean) : string`
`function GetTempDir : string`

Visibility: default

Description: `GetTempDir` returns the temporary directory of the system. If `Global` is `True` (the default value) it returns the system temporary directory, if it is `False` then a directory private to the user is returned. The returned name will end with a directory delimiter character.

These directories may be the same. No guarantee is made that this directory exists or is writeable by the user.

The `OnGetTempDir` (1645) handler may be set to provide custom handling of this routine: One could implement callbacks which take into consideration frameworks like KDE or GNOME, and return a different value from the default system implementation.

Errors: On error, an empty string is returned.

See also: `OnGetTempDir` (1645), `GetTempFileName` (1733)

76.15.163 GetTempFileName

Synopsis: Return the name of a temporary file.

Declaration:

```
function GetTempFileName(const Dir: string; const Prefix: string)
                        : string
function GetTempFileName : string
function GetTempFileName(Dir: PAnsiChar; Prefix: PAnsiChar;
                        uUnique: DWord; TempFileName: PAnsiChar) : DWord
```

Visibility: default

Description: `GetTempFileName` returns the name of a temporary file in directory `Dir`. The name of the file starts with `Prefix`.

If `Dir` is empty, the value returned by `GetTempDir` is used, and if `Prefix` is empty, 'TMP' is used.

The `OnGetTempFile` (1645) handler may be set to provide custom handling of this routine: One could implement callbacks which take into consideration frameworks like KDE or GNOME, and return a different value from the default system implementation.

Errors: On error, an empty string is returned.

See also: `GetTempDir` (1732), `OnGetTempFile` (1645)

76.15.164 GetTickCount

Synopsis: Get tick count (32-bit, deprecated).

Declaration:

```
function GetTickCount : LongWord
```

Visibility: default

Description: `GetTickCount` returns an increasing clock tick count in milliseconds. It is useful for time measurements, but no assumptions should be made as to the interval between the ticks. This function is provided for Delphi compatibility, use `GetTickCount64` (1734) instead.

See also: `GetTickCount64` (1734), `Now` (1745), `Time` (1781), `Sleep` (1751)

76.15.165 GetTickCount64

Synopsis: Get tick count (64-bit).

Declaration: `function GetTickCount64 : QWord`

Visibility: default

Description: `GetTickCount64` returns an increasing clock tick count in milliseconds. It is useful for time measurements, but no assumptions should be made as to the interval between the ticks.

See also: [Now \(1745\)](#), [Time \(1781\)](#), [Sleep \(1751\)](#)

76.15.166 GetUniversalTime

Declaration: `function GetUniversalTime(var SystemTime: TSystemTime) : Boolean`

Visibility: default

76.15.167 GetUserDir

Synopsis: Returns the current user's home directory.

Declaration: `function GetUserDir : string`

Visibility: default

Description: `GetUserDir` returns the home directory of the current user. On Unix-like systems (that includes Mac OS X), this is the value of the HOME environment variable. On Windows, this is the PROFILE special folder. On all other platforms, the application installation directory is returned.

If non-empty, it contains a trailing path delimiter.

See also: [GetAppConfigDir \(1726\)](#)

76.15.168 GuidCase

Synopsis: Return the index of a GUID in an array of GUID values.

Declaration: `function GuidCase(const GUID: TGuid; const List: Array of TGuid)
: Integer`

Visibility: default

Description: `GuidCase` returns the index of GUID in the array `List`, where 0 denotes the first element in the list. If GUID is not present in the list, -1 is returned.

See also: [IsEqualGUID \(1739\)](#)

76.15.169 GUIDToString

Synopsis: Convert a TGUID to a string representation.

Declaration: `function GUIDToString(const GUID: TGuid) : string`

Visibility: default

Description: `GUIDToString` converts the GUID identifier in `GUID` to a string representation in the form

```
{XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}
```

Where each X is a hexadecimal digit.

Errors: None.

See also: Supports ([1779](#)), #rtl.system.TGUID ([1583](#)), StringToGUID ([1759](#)), IsEqualGuid ([1739](#))

76.15.170 HashName

Synopsis: Calculate a hash from a null-terminated string.

Declaration: `function HashName(Name: PAnsiChar) : LongWord`

Visibility: default

Description: `HashName` calculates a hash value from a null terminated string. The hash value is calculated in such a way that it returns the same value for strings that only differ in case.

76.15.171 HookSignal

Synopsis: Hook a specified signal.

Declaration: `procedure HookSignal(RtlSigNum: Integer)`

Visibility: default

Description: `HookSignal` installs the RTL default signal handler for signal `RtlSigNum`. It does not check whether the signal is already handled, and should therefor only be called if `InquireSignal` returns `ssNotHooked`.

76.15.172 IfThen

Synopsis: Return one of two values depending on boolean condition.

Declaration: `function IfThen<T>(val: Boolean; const iftrue: T; const iffalse: T) : T
; Overload`

Visibility: default

Description: `IfThen` returns `iftrue` if `val` is `True`, `iffalse` if it is `False`. Note that both arguments will always be evaluated, so this does not behave like the conditional as found in some other languages.

76.15.173 IncAMonth

Synopsis: Increase a date with a certain amount of months.

Declaration: `procedure IncAMonth(var Year: Word; var Month: Word; var Day: Word;
NumberOfMonths: Integer)`

Visibility: default

Description: `IncAMonth` increases the date as specified by `Year`, `Month`, `Day` with `NumberOfMonths`. It takes care of the number of days in a month when calculating the result.

This function does the same as `IncMonth` ([1737](#)), but operates on an already decoded date.

See also: `IncMonth` ([1737](#))

76.15.174 IncludeLeadingPathDelimiter

Synopsis: Prepend a path delimiter if there is not already one.

Declaration: `function IncludeLeadingPathDelimiter(const Path: UnicodeString)
: UnicodeString
function IncludeLeadingPathDelimiter(const Path: RawByteString)
: RawByteString`

Visibility: default

Description: `IncludeLeadingPathDelimiter` will insert a path delimiter (`#rtl.system.DirectorySeparator` (1344)) in the first position of `Path`, if there is not already a directory separator at that position. It will return the resulting string. If the path is empty, a `DirectorySeparator` character is returned.

See also: `IncludeTrailingPathDelimiter` (1736), `ExcludeLeadingPathDelimiter` (1686), `ExcludeTrailingPathDelimiter` (1687), `ConcatPaths` (1671)

Listing: `./examples/sysutex/ex94.pp`

Program Example94;

{ This program demonstrates the IncludeLeadingPathDelimiter function }

Uses sysutils;

Begin

End.

76.15.175 IncludeTrailingBackslash

Synopsis: Add trailing directory separator to a pathname, if needed.

Declaration: `function IncludeTrailingBackslash(const Path: UnicodeString)
: UnicodeString
function IncludeTrailingBackslash(const Path: RawByteString)
: RawByteString`

Visibility: default

Description: `IncludeTrailingBackslash` is provided for backwards compatibility with Delphi. Use `IncludeTrailingPathDelimiter` (1736) instead.

See also: `IncludeTrailingPathDelimiter` (1736), `ExcludeTrailingPathDelimiter` (1687), `PathDelim` (1622), `IsPathDelimiter` (1741)

76.15.176 IncludeTrailingPathDelimiter

Synopsis: Add trailing directory separator to a pathname, if needed.

Declaration: `function IncludeTrailingPathDelimiter(const Path: UnicodeString)
: UnicodeString
function IncludeTrailingPathDelimiter(const Path: RawByteString)
: RawByteString`

Visibility: default

Description: `IncludeTrailingPathDelimiter` adds a trailing path delimiter character (`PathDelim` (1622)) to `Path` if none is present yet, and returns the result.

If `Path` is empty, a path delimiter is returned, for Delphi compatibility.

See also: `IncludeTrailingBackslash` (1736), `ExcludeTrailingPathDelimiter` (1687), `PathDelim` (1622), `IsPathDelimiter` (1741)

76.15.177 IncMonth

Synopsis: Increases the month in a `TDateTime` value with a given amount.

Declaration: `function IncMonth(const DateTime: TDateTime; NumberOfMonths: Integer) : TDateTime`

Visibility: default

Description: `IncMonth` increases the month number in `DateTime` with `NumberOfMonths`. It wraps the result as to get a month between 1 and 12, and updates the year accordingly. `NumberOfMonths` can be negative, and can be larger than 12 (in absolute value).

Errors: None.

See also: `Date` (1673), `Time` (1781), `Now` (1745)

Listing: `./examples/sysutex/ex15.pp`

Program Example15;

{ This program demonstrates the IncMonth function }

Uses sysutils;

Var ThisDay : TDateTime;

Begin

 ThisDay:=Date;

Writeln ('ThisDay : ',DateToStr(ThisDay));

Writeln ('6 months ago :',DateToStr(IncMonth(ThisDay,-6)));

Writeln ('6 months from now :',DateToStr(IncMonth(ThisDay,6)));

Writeln ('12 months ago :',DateToStr(IncMonth(ThisDay,-12)));

Writeln ('12 months from now :',DateToStr(IncMonth(ThisDay,12)));

Writeln ('18 months ago :',DateToStr(IncMonth(ThisDay,-18)));

Writeln ('18 months from now :',DateToStr(IncMonth(ThisDay,18)));

End.

76.15.178 InquireSignal

Synopsis: Check whether a signal handler is set (unix only).

Declaration: `function InquireSignal(RtlSigNum: Integer) : TSignalState`

Visibility: default

Description: `RtlSigNum` will check whether the signal `RtlSigNum` is being handled, and by whom. It returns a `TSignalState` result to report the state of the signal, which can be one of the following values:

ssNotHooked No signal handler is set for the signal.

ssHookedA signal handler is set by the RTL code for the signal.

ssOverriddenA signal handler was set for the signal by third-party code.

This routine works by resetting the signal handlers, so it is risky to call.

76.15.179 IntToHex

Synopsis: Convert an integer value to a hexadecimal string.

Declaration:

```
function IntToHex(Value: LongInt; Digits: Integer) : string
function IntToHex(Value: Int64; Digits: Integer) : string
function IntToHex(Value: QWord; Digits: Integer) : string
function IntToHex(Value: Int8) : string
function IntToHex(Value: UInt8) : string
function IntToHex(Value: Int16) : string
function IntToHex(Value: UInt16) : string
function IntToHex(Value: Int32) : string
function IntToHex(Value: UInt32) : string
function IntToHex(Value: Int64) : string
function IntToHex(Value: UInt64) : string
```

Visibility: default

Description: `IntToHex` converts `Value` to a hexadecimal string representation. The result will contain at least `Digits` characters. If `Digits` is less than the needed number of characters, the string will NOT be truncated. If `Digits` is larger than the needed number of characters, the result is padded with zeroes.

Errors: None.

See also: `IntToStr` ([1738](#))

Listing: `./examples/sysutex/ex73.pp`

Program Example73;

{ This program demonstrates the IntToHex function }

Uses sysutils;

Var I : longint;

Begin

For I:=0 to 31 **do**

begin

Writeln (IntToHex(1 shl I,8));

Writeln (IntToHex(15 shl I,8))

end;

End.

76.15.180 IntToStr

Synopsis: Convert an integer value to a decimal string.

Declaration: `function IntToStr(Value: LongInt) : string`
`function IntToStr(Value: Int64) : string`
`function IntToStr(Value: QWord) : string`

Visibility: default

Description: `IntToStr` converts `Value` to its string representation. The resulting string has only as much characters as needed to represent the value. If the value is negative a minus sign is prepended to the string.

Errors: None.

See also: `IntToHex` ([1738](#)), `StrToInt` ([1773](#)), `UIntToStr` ([1791](#))

Listing: `./examples/sysutex/ex74.pp`

Program Example74;

{ This program demonstrates the IntToStr function }

Uses sysutils;

Var I : longint;

Begin

For I:=0 to 31 do

begin

Writeln (IntToStr(1 shl I));

Writeln (IntToStr(15 shl I));

end;

End.

76.15.181 IsDelimiter

Synopsis: Check whether a given string is a delimiter character.

Declaration: `function IsDelimiter(const Delimiters: string; const S: string;`
`Index: SizeInt) : Boolean`

Visibility: default

Description: `IsDelimiter` checks whether the `Index`-th character in the string `S` is a delimiter character as passed in `Delimiters`. If `Index` is out of range, `False` is returned.

Errors: None.

See also: `LastDelimiter` ([1742](#))

76.15.182 IsEqualGUID

Synopsis: Check whether two `TGUID` variables are equal.

Declaration: `function IsEqualGUID(const guid1: TGuid; const guid2: TGuid) : Boolean`

Visibility: default

Description: `IsEqualGUID` checks whether `guid1` and `guid2` are equal, and returns `True` if this is the case, or `False` otherwise.

See also: `Supports` ([1779](#)), `#rtl.system.TGUID` ([1583](#)), `StringToGUID` ([1759](#)), `GuidToString` ([1734](#))

76.15.183 IsFileNameCasePreserving

Declaration: `function IsFileNameCasePreserving(const aFileName: RawByteString)
: Boolean
function IsFileNameCasePreserving(const aFileName: UnicodeString)
: Boolean`

Visibility: default

76.15.184 IsFileNameCaseSensitive

Declaration: `function IsFileNameCaseSensitive(const aFileName: RawByteString)
: Boolean
function IsFileNameCaseSensitive(const aFileName: UnicodeString)
: Boolean`

Visibility: default

76.15.185 IsLeadChar

Synopsis: Is the character code the lead character of a multi-byte character.

Declaration: `function IsLeadChar(C: AnsiChar) : Boolean; Overload
function IsLeadChar(B: Byte) : Boolean; Overload
function IsLeadChar(Ch: WideChar) : Boolean; Overload`

Visibility: default

Description: `IsLeadChar` returns `True` if the character `C` (or byte `B`) is the lead character (or byte) of a multi-byte character. In case of `Ch` a UTF-32 lead character.

Errors: None.

76.15.186 IsLeapYear

Synopsis: Determine whether a year is a leap year.

Declaration: `function IsLeapYear(Year: Word) : Boolean`

Visibility: default

Description: `IsLeapYear` returns `True` if `Year` is a leap year, `False` otherwise.

Errors: None.

See also: `IncMonth` ([1737](#)), `Date` ([1673](#))

Listing: `./examples/sysutex/ex16.pp`

Program Example16;

{ This program demonstrates the IsLeapYear function }

Uses sysutils;

Var YY,MM,dd : Word;

Procedure TestYear (Y : Word);

```
begin
  Writeln (Y,' is leap year : ',IsLeapYear(Y));
end;
```

```
Begin
  DeCodeDate(Date,YY,mm,dd);
  TestYear(yy);
  TestYear(2000);
  TestYear(1900);
  TestYear(1600);
  TestYear(1992);
  TestYear(1995);
End.
```

76.15.187 IsPathDelimiter

Synopsis: Is the character at the given position a pathdelimiter ?

```
Declaration: function IsPathDelimiter(const Path: UnicodeString; Index: Integer)
              : Boolean
              function IsPathDelimiter(const Path: RawByteString; Index: Integer)
              : Boolean
```

Visibility: default

Description: `IsPathDelimiter` returns `True` if the character at position `Index` equals `PathDelim` ([1622](#)), i.e. if it is a path delimiter character for the current platform.

Errors: `IncludeTrailingPathDelimiter` ([1736](#))`ExcludeTrailingPathDelimiter` ([1687](#))`PathDelim` ([1622](#))

76.15.188 IsValidIdent

Synopsis: Check whether a string is a valid identifier name.

```
Declaration: function IsValidIdent(const Ident: string; AllowDots: Boolean;
              StrictDots: Boolean) : Boolean
```

Visibility: default

Description: `IsValidIdent` returns `True` if `Ident` can be used as a component name. It returns `False` otherwise. `Ident` must consist of a letter or underscore, followed by a combination of letters, numbers or underscores to be a valid identifier.

Errors: None.

Listing: `./examples/sysutex/ex75.pp`

Program Example75;

{ This program demonstrates the IsValidIdent function }

Uses sysutils;

Procedure Testit (S : **String**);

begin

```

Write ('"',S,'" is ');
If not IsValidIdent(S) then
  Write('NOT ');
  Writeln ('a valid identifier');
end;

Begin
  Testit ('_MyObj');
  Testit ('My__Obj1');
  Testit ('My_1_Obj');
  Testit ('1MyObject');
  Testit ('My@Object');
  Testit ('M123');
End.

```

76.15.189 LastDelimiter

Synopsis: Return the last occurrence of a set of delimiters in a string.

Declaration: `function LastDelimiter(const Delimiters: string; const S: string) : SizeInt`

Visibility: default

Description: `LastDelimiter` returns the *last* occurrence of any character in the set `Delimiters` in the string `S`.

Listing: ./examples/sysutex/ex88.pp

```

Program example88;

{ This program demonstrates the LastDelimiter function }

uses SysUtils;

begin
  Writeln(LastDelimiter("\.:",'c:\filename.ext'));
end.

```

76.15.190 LeftStr

Synopsis: Return a number of characters starting at the left of a string.

Declaration: `function LeftStr(const S: string; Count: Integer) : string`

Visibility: default

Description: `LeftStr` returns the `Count` leftmost characters of `S`. It is equivalent to a call to `Copy(S, 1, Count)`.

Errors: None.

See also: `RightStr` ([1748](#)), `TrimLeft` ([1784](#)), `TrimRight` ([1785](#)), `Trim` ([1783](#))

Listing: ./examples/sysutex/ex76.pp

Program Example76;*{ This program demonstrates the LeftStr function }***Uses** sysutils;**Begin****Writeln** (LeftStr('abcdefghijklmnopqrstuvwxyz',20));**Writeln** (LeftStr('abcdefghijklmnopqrstuvwxyz',15));**Writeln** (LeftStr('abcdefghijklmnopqrstuvwxyz',1));**Writeln** (LeftStr('abcdefghijklmnopqrstuvwxyz',200));**End.**

76.15.191 ListIndexError**Declaration:** procedure ListIndexError(aIndex: Integer; aMax: Integer; aObj: TObject)

Visibility: default

76.15.192 ListIndexErrorMsg**Declaration:** function ListIndexErrorMsg(aIndex: SizeInt; aMaxIndex: SizeInt;
const aListObjName: string) : string
; Overload
function ListIndexErrorMsg(aIndex: SizeInt; aMaxIndex: SizeInt;
aListObj: TObject) : string; Overload

Visibility: default

76.15.193 LoadStr

Synopsis: Load a string from the resource tables.

Declaration: function LoadStr(Ident: Integer) : string

Visibility: default

Description: This function is not yet implemented. resources are not yet supported.

76.15.194 LocalTimeToUniversal**Declaration:** function LocalTimeToUniversal(LT: TDateTime) : TDateTime
function LocalTimeToUniversal(LT: TDateTime; TZOffset: Integer)
: TDateTime

Visibility: default

76.15.195 LowerCase

Synopsis: Return a lowercase version of a string.


```
Declaration: function LowerCase(const s: ansistring) : ansistring; Overload
function LowerCase(const s: ansistring; LocaleOptions: TLocaleOptions)
: ansistring; Overload
function LowerCase(const V: variant) : string; Overload
function LowerCase(const s: UnicodeString) : UnicodeString; Overload
```

Visibility: default

Description: LowerCase returns the lowercase equivalent of S. Ansi characters are not taken into account, only ASCII codes below 127 are converted. It is completely equivalent to the lowercase function of the system unit, and is provided for compatibility only.

Lowercase does not change the number of characters (or bytes) in an ansistring or shortstring.

Errors: None.

See also: [AnsiLowerCase \(1653\)](#), [UpperCase \(1795\)](#), [AnsiUpperCase \(1661\)](#)

Listing: ./examples/sysutex/ex77.pp

Program Example77;

```
{ This program demonstrates the LowerCase function }
```

Uses sysutils;

Begin

```
WriteLn (LowerCase('THIS WILL COME out all LoWeRcAsE !'));
```

End.

76.15.196 MSecsToTimeStamp

Synopsis: Convert a number of milliseconds to a `DateTime` value.

```
Declaration: function MSecsToTimeStamp(MSecs: Comp) : TTimeStamp
              function MSecsToTimeStamp(MSecs: Int64) : TTimeStamp
```

Visibility: default

Description: MSecsToTimeStamp converts the given number of milliseconds to a TTimeStamp date/time notation.

Use `TTimeStamp` variables if you need to keep very precise track of time.

Errors: None.

See also: [TimeStampToMSecs \(1783\)](#), [DateTimeToTimeStamp \(1677\)](#)

Listing: ./examples/sysutex/ex17.pp

Program Example17;

{ This program demonstrates the MSecsToTimeStamp function }

Uses sysutils;

```
Var MS : Comp;
```

TS : TTimeStamp;

DT : TDateTime;

Begin

```

TS:=DateTimeToTimeStamp(Now);
Writeln ('Now in days since 1/1/0001 : ',TS.Date);
Writeln ('Now in millisecs since midnight : ',TS.Time);
MS:=TimeStampToMSecs(TS);
Writeln ('Now in millisecs since 1/1/0001 : ',MS);
MS:=MS-1000*3600*2;
TS:=MSecsToTimeStamp(MS);
DT:=TimeStampToDateTime(TS);
Writeln ('Now minus 1 day : ',DateTimeToStr(DT));

```

End.**76.15.197 NewStr**

Synopsis: Allocate a new ansistring on the heap.

Declaration: `function NewStr(const S: string) : PString; Overload`

Visibility: default

Description: `NewStr` assigns a new dynamic string on the heap, copies `S` into it, and returns a pointer to the newly assigned string.

This function is obsolete, and shouldn't be used any more. The `AnsiString` mechanism also allocates ansistrings on the heap, and should be preferred over this mechanism.

For an example, see `AssignStr` (1663).

Errors: If not enough memory is present, an `EOutOfMemory` exception will be raised.

See also: `AssignStr` (1663), `DisposeStr` (1682)

76.15.198 Now

Synopsis: Returns the current date and time.

Declaration: `function Now : TDateTime`

Visibility: default

Description: `Now` returns the current date and time. It is equivalent to `Date+Time`.

Errors: None.

See also: `Date` (1673), `Time` (1781)

Listing: `./examples/sysutex/ex18.pp`

Program Example18;

{ This program demonstrates the Now function }

Uses sysutils;

Begin

```

Writeln ('Now : ',DateTimeToStr(Now));

```

End.

76.15.199 NowUTC

Declaration: `function NowUTC : TDateTime`

Visibility: default

76.15.200 OutOfMemoryError

Synopsis: Raise an `EOutOfMemory` exception.

Declaration: `procedure OutOfMemoryError`

Visibility: default

Description: `OutOfMemoryError` raises an `EOutOfMemory` (1818) exception, with an exception object that has been allocated on the heap at program startup. The program should never create an `EOutOfMemory` (1818) exception, but always call this routine.

See also: `EOutOfMemory` (1818)

76.15.201 QuotedStr

Synopsis: Return a quotes version of a string.

Declaration: `function QuotedStr(const S: string) : string`

Visibility: default

Description: `QuotedStr` returns the string `S`, quoted with single quotes. This means that `S` is enclosed in single quotes, and every single quote in `S` is doubled. It is equivalent to a call to `AnsiQuotedStr(S, '"')`.

Errors: None.

See also: `AnsiQuotedStr` (1654), `AnsiExtractQuotedStr` (1651)

Listing: `./examples/sysutex/ex78.pp`

Program Example78;

{ This program demonstrates the QuotedStr function }

Uses sysutils;

Var S : AnsiString;

Begin

 S:='He said "Hello" and walked on';

Writeln (S);

Writeln (' becomes');

Writeln (QuotedStr(S));

End.

76.15.202 RaiseLastError

Synopsis: Raise an exception with the last Operating System error code.

Declaration: `procedure RaiseLastError; Overload`
`procedure RaiseLastError(LastError: Integer); Overload`

Visibility: default

Description: `RaiseLastError` raises an `EOSError` (1818) exception with the error code returned by `GetLastError`. If the Error code is nonzero, then the corresponding error message will be returned. If the error code is zero, a standard message will be returned.

Errors: This procedure may not be implemented on all platforms. If it is not, then a normal Exception (1821) will be raised.

See also: `EOSError` (1818), `GetLastError` (1731), Exception (1821)

76.15.203 RangeIndexError

Declaration: `procedure RangeIndexError(aIndex: SizeInt; aMaxIndex: SizeInt;`
`aListObj: TObject)`

Visibility: default

76.15.204 RemoveDir

Synopsis: Remove a directory from the file system.

Declaration: `function RemoveDir(const Dir: RawByteString) : Boolean`
`function RemoveDir(const Dir: UnicodeString) : Boolean`

Visibility: default

Description: `RemoveDir` removes directory `Dir` from the disk. If the directory is not absolute, it is appended to the current working directory.

For an example, see `CreateDir` (1671).

Errors: In case of error (e.g. the directory isn't empty) the function returns `False`. If successful, `True` is returned.

76.15.205 RenameFile

Synopsis: Rename a file.

Declaration: `function RenameFile(const OldName: UnicodeString;`
`const NewName: UnicodeString) : Boolean`
`function RenameFile(const OldName: RawByteString;`
`const NewName: RawByteString) : Boolean`

Visibility: default

Description: `RenameFile` renames a file from `OldName` to `NewName`. The function returns `True` if successful, `False` otherwise. For safety, the new name must be a full path specification, including the directory, otherwise it will be assumed to be a filename relative to the current working directory. *Remark:* The implementation of `RenameFile` relies on the underlying OS's support for renaming/moving a file. Whether or not a file can be renamed across disks or partitions depends entirely on

the OS. On unix-like OS-es, the rename function will fail when used across partitions. On Windows, it will work.

Errors: On Error, `False` is returned.

See also: `DeleteFile` ([1680](#))

Listing: `./examples/sysutex/ex44.pp`

Program `Example44`;

{ This program demonstrates the `RenameFile` function }

Uses `sysutils`;

Var `F` : `Longint`;
 `S` : `String`;

Begin

`S:= 'Some short file.';`
 `F:= FileCreate ('test.dap');`
 `FileWrite(F,S[1],Length(S));`
 `FileClose(F);`
 If `RenameFile ('test.dap','test.dat')` **then**
 `Writeln ('Successfully renamed files.');`

End.

76.15.206 ReplaceDate

Synopsis: Replace the date part of a date/time stamp.

Declaration: `procedure ReplaceDate(var DateTime: TDateTime; const NewDate: TDateTime)`

Visibility: `default`

Description: `ReplaceDate` replaces the date part of `DateTime` with `NewDate`. The time part is left unchanged.

See also: `ReplaceTime` ([1748](#))

76.15.207 ReplaceTime

Synopsis: Replace the time part.

Declaration: `procedure ReplaceTime(var dateTime: TDateTime; NewTime: TDateTime)`

Visibility: `default`

Description: `ReplaceTime` replaces the time part in `dateTime` with `NewTime`. The date part remains untouched.

76.15.208 RightStr

Synopsis: Return a number of characters from a string, starting at the end.

Declaration: `function RightStr(const S: string; Count: Integer) : string`

Visibility: `default`

Description: `RightStr` returns the `Count` rightmost characters of `S`. It is equivalent to a call to `Copy (S, Length (S) + 1 - Count, Count, S)`.
If `Count` is larger than the actual length of `S` only the real length will be used.

Errors: None.

See also: `LeftStr` ([1742](#)), `Trim` ([1783](#)), `TrimLeft` ([1784](#)), `TrimRight` ([1785](#))

Listing: `./examples/sysutex/ex79.pp`

Program Example79;

{ This program demonstrates the RightStr function }

Uses sysutils;

Begin

Writeln (`RightStr('abcdefghijklmnopqrstuvwxyz',20)`);

Writeln (`RightStr('abcdefghijklmnopqrstuvwxyz',15)`);

Writeln (`RightStr('abcdefghijklmnopqrstuvwxyz',1)`);

Writeln (`RightStr('abcdefghijklmnopqrstuvwxyz',200)`);

End.

76.15.209 SafeFormat

Declaration: `function SafeFormat(const Fmt: AnsiString; const Args: Array of const; const FormatSettings: TFormatSettings) : UTF8String; Overload`
`function SafeFormat(const Fmt: AnsiString; const Args: Array of const) : UTF8String; Overload`

Visibility: default

76.15.210 SafeLoadLibrary

Synopsis: Load a library safely.

Declaration: `function SafeLoadLibrary(const FileName: AnsiString; ErrorMode: DWord) : HMODULE`

Visibility: default

Description: `SafeLoadLibrary` saves and restores some registers before and after issuing a call to `LoadLibrary`.

Errors: None.

76.15.211 SameFileName

Synopsis: Are two filenames referring to the same file ?

Declaration: `function SameFileName(const S1: string; const S2: string) : Boolean`

Visibility: default

Description: `SameFileName` returns `True` if calling `AnsiCompareFileName` ([1649](#)) with arguments `S1` and `S2` returns 0, and returns `False` otherwise.

Errors: None.

See also: `AnsiCompareFileName` ([1649](#))

76.15.212 SameStr

Synopsis: Check whether 2 strings are the same, case insensitive.

```
Declaration: function SameStr(const s1: string; const s2: string) : Boolean
                ; Overload
function SameStr(const s1: string; const s2: string;
                LocaleOptions: TLocaleOptions) : Boolean; Overload
```

Visibility: default

Description: SameStr checks whether S1 and S2 are the same. This is equivalent to checking that CompareStr (1668) returns 0.

Errors: None.

See also: [CompareStr \(1668\)](#)

76.15.213 SameText

Synopsis: Checks whether 2 strings are the same (case insensitive).

```
Declaration: function SameText(const s1: string; const s2: string) : Boolean
                ; Overload
    function SameText(const s1: string; const s2: string;
                        LocaleOptions: TLocaleOptions) : Boolean; Overload
```

Visibility: default

Description: SameText calls CompareText (1669) with S1 and S2 as parameters and returns True if the result of that call is zero, or False otherwise.

Errors: None.

See also: [CompareText \(1669\)](#), [AnsiSameText \(1654\)](#), [AnsiSameStr \(1654\)](#)

76.15.214 SetCurrentDir

Synopsis: Set the current directory of the application.

```
Declaration: function SetCurrentDir(const NewDir: RawByteString) : Boolean
              function SetCurrentDir(const NewDir: UnicodeString) : Boolean
```

Visibility: default

Description: `SetCurrentDir` sets the current working directory of your program to `NewDir`. It returns `True` if the function was successful, `False` otherwise.

Errors: In case of error, `False` is returned.

See also: [GetCurrentDir \(1727\)](#)

76.15.215 SetDirSeparators

Synopsis: Set the directory separators to the known directory separators.

Declaration: `function SetDirSeparators(const FileName: UnicodeString) : UnicodeString`
`function SetDirSeparators(const FileName: RawByteString) : RawByteString`

Visibility: default

Description: `SetDirSeparators` returns `FileName` with all possible `DirSeparators` replaced by `OSDirSeparator`.

Errors: None.

See also: `ExpandFileName` (1688), `ExtractFilePath` (1692), `ExtractFileDir` (1690)

Listing: `./examples/sysutex/ex47.pp`

Program Example47;

{ This program demonstrates the SetDirSeparators function }

Uses sysutils;

Begin

`WriteLn (SetDirSeparators('/pp/bin/win32\ppc386'));`

End.

76.15.216 ShowException

Synopsis: Show the current exception to the user.

Declaration: `procedure ShowException(ExceptObject: TObject; ExceptAddr: Pointer)`

Visibility: default

Description: `ShowException` shows a message stating that a `ExceptObject` was raised at address `ExceptAddr`.

It uses `ExceptionErrorMessage` (1685) to create the message, and is aware of the fact whether the application is a console application or a GUI application. For a console application, the message is written to standard error output. For a GUI application, `OnShowException` (1645) is executed.

Errors: If, for a GUI application, `OnShowException` (1645) is not set, no message will be displayed to the user.

The exception message can be at most 255 characters long: It is possible that no memory can be allocated on the heap, so ansistrings are not available, so a shortstring is used to display the message.

See also: `ExceptObject` (1685), `ExceptAddr` (1684), `ExceptionErrorMessage` (1685)

76.15.217 Sleep

Synopsis: Suspend execution of a program for a certain time.

Declaration: `procedure Sleep(milliseconds: Cardinal)`

Visibility: default

Description: `Sleep` suspends the execution of the program for the specified number of milliseconds (`milliseconds`).

After the specified period has expired, program execution resumes.

Remark The indicated time is not exact, i.e. it is a minimum time. No guarantees are made as to the exact duration of the suspension.

76.15.218 SScanf

Synopsis: Scan a string for substrings and return the content of these substrings as typed values.

Declaration: `function SScanf(const s: string; const fmt: string;
const Pointers: Array of Pointer) : Integer`

Visibility: default

Description: `SScanF` does in essence the opposite of `Format` (1715): it scans the string `S` for the elements specified in `Fmt`, and returns the value of the found elements in the memory locations pointed to by the addresses in `Pointers`. The `Fmt` can contain placeholders of the form `%X` where `X` can be one of the following characters:

- d** Placeholder for a decimal number.
- f** Placeholder for a floating point number (an extended)
- s** Placeholder for a string of arbitrary length.
- c** Placeholder for a single character

The `Pointers` array contains a list of pointers, each pointer should point to a memory location of a type that corresponds to the type of placeholder in that position:

- d** A pointer to an integer.
- f** A pointer to an extended.
- s** A pointer to an ansistring.
- c** A pointer to a single character.

On return, these locations will be filled with the actual values found in `S` for the placeholders in `fmt`. The return value of the function is the number of found values.

Errors: No error checking is performed on the type of the memory location.

See also: `Format` (1715)

Listing: `./examples/sysutex/ex98.pp`

```
Program Example98;
{$mode objfpc}
{$h+}
{ This program demonstrates the function }

Uses sysutils;

var
  count,i: Integer;
  f: Extended;
  s: String;

begin
  count:=SScanf('234 32.4 hello','%d %f %s',[@i,@f,@s]);
  writeln(count,' ',i,' ',f,' ',s);
End.
```

76.15.219 StrAlloc

Synopsis: Allocate a null-terminated string on the heap.

Declaration: `function StrAlloc(Size: Cardinal) : PAnsiChar`

Visibility: default

Description: `StrAlloc` reserves memory on the heap for a string with length `Len`, terminating `#0` included, and returns a pointer to it.

Additionally, `StrAlloc` allocates 4 extra bytes to store the size of the allocated memory. Therefore this function is NOT compatible with the `StrAlloc` (1283) function of the `Strings` unit.

For an example, see `StrBufSize` (1753).

Errors: None.

See also: `StrBufSize` (1753), `StrDispose` (1756), `StrAlloc` (1283)

76.15.220 StrBufSize

Synopsis: Return the size of a null-terminated string allocated on the heap.

Declaration: `function StrBufSize(Str: PAnsiChar) : Cardinal`
`function StrBufSize(str: PWideChar) : Cardinal`

Visibility: default

Description: `StrBufSize` returns the memory allocated for `Str`. This function ONLY gives the correct result if `Str` was allocated using `StrAlloc` (1753).

Errors: If no more memory is available, a runtime error occurs.

See also: `StrAlloc` (1753), `StrDispose` (1756)

Listing: `./examples/sysutex/ex46.pp`

Program Example46;

{ This program demonstrates the StrBufSize function }
{ \$H+ }

Uses sysutils;

Const S = 'Some nice string';

Var P : Pchar;

Begin

 P:=StrAlloc(Length(S)+1);
 StrPCopy(P,S);
 Write (P, ' has length ',length(S));
 Writeln (' and buffer size ',StrBufSize(P));
 StrDispose(P);

End.

76.15.221 StrByteType

Synopsis: Return the type of byte in a null-terminated string for a multi-byte character set.

Declaration: `function StrByteType(Str: PAnsiChar; Index: SizeUInt) : TmbcsByteType`

Visibility: default

Description: `StrByteType` returns the type of byte in the null-terminated string `Str` at (0-based) position `Index`.

Errors: No checking on the index is performed.

See also: `TmbcsByteType` ([1635](#)), `ByteType` ([1666](#))

76.15.222 strcat

Synopsis: Concatenate 2 null-terminated strings.

Declaration: `function StrCat(dest: PAnsiChar; source: PAnsiChar) : PAnsiChar`
`function strcat(dest: PWideChar; source: PWideChar) : PWideChar`

Visibility: default

Description: Attaches `Source` to `Dest` and returns `Dest`.

Errors: No length checking is performed.

See also: `StrLCat` ([1760](#))

Listing: `./examples/stringex/ex11.pp`

Program Example11;

Uses strings;

{ Program to demonstrate the StrCat function. }

Const P1 : PChar = 'This is a PChar String.';

Var P2 : PChar;

begin

 P2:=StrAlloc (StrLen(P1)*2+1);
 StrMove (P2,P1,StrLen(P1)+1); { P2=P1 }
 StrCat (P2,P1); { Append P2 once more }
 Writeln ('P2 : ',P2);
 StrDispose(P2);

end.

76.15.223 StrCharLength

Synopsis: Return the length of a null-terminated string in characters.

Declaration: `function StrCharLength(const Str: PAnsiChar) : SizeInt`

Visibility: default

Description: `StrCharLength` returns the length of the null-terminated string `Str` (a widestring) in characters (not in bytes). It uses the widestring manager to do this.

76.15.224 strcmp

Synopsis: Compare 2 null-terminated strings, case sensitive.

```
Declaration: function StrComp(str1: PAnsiChar; str2: PAnsiChar) : SizeInt
             function strcmp(str1: PWideChar; str2: PWideChar) : SizeInt
```

Visibility: default

Description: Compares the null-terminated strings S1 and S2. The result is

- A negative Longint when $S1 < S2$.
- 0 when $S1 = S2$.
- A positive Longint when $S1 > S2$.

For an example, see `StrLComp` ([1760](#)).

Errors: None.

See also: [StrLComp \(1760\)](#), [StrIComp \(1758\)](#), [StrLIComp \(1763\)](#)

76.15.225 StrCopy

Synopsis: Copy a null-terminated string.

```
Declaration: function StrCopy(dest: PAnsiChar; source: PAnsiChar) : PAnsiChar
              ; Overload
function StrCopy(Dest: PWideChar; Source: PWideChar) : PWideChar
              ; Overload
```

Visibility: default

Description: Copy the null terminated string in *Source* to *Dest*, and returns a pointer to *Dest*. *Dest* needs enough room to contain *Source*, i.e. `StrLen (Source) + 1` bytes.

Errors: No length checking is performed.

See also: StrPCopy (1766), StrLCopy (1761), StrECopy (1756)

Listing: ./examples/stringex/ex4.pp

Program Example4;

Uses strings;

```
{ Program to demonstrate the StrCopy function. }
```

```
Const P : PChar = 'This is a PCHAR string.';
```

```
var PP : PChar;
```

begin

```

PP:=StrAlloc(Strlen(P)+1);
StrCopy (PP,P);
If StrComp (PP,P)<>0 then
    Writeln ('Oh-oh problems...')
else
    Writeln ('All is well : PP=',PP);
StrDispose(PP);
end.

```

76.15.226 StrDispose

Synopsis: Dispose of a null-terminated string on the heap.

Declaration: `procedure StrDispose(Str: PAnsiChar)`
`procedure StrDispose(str: PWideChar)`

Visibility: default

Description: `StrDispose` frees any memory allocated for `Str`. This function will only function correctly if `Str` has been allocated on the heap, for example using `StrAlloc` (1753) or `StrNew` (1764) from the `SysUtils` unit.

For an example, see `StrBufSize` (1753).

Errors: If an invalid pointer is passed, or a pointer not allocated with `StrAlloc`, an error may occur.

See also: `StrBufSize` (1753), `StrAlloc` (1753), `StrNew` (1764)

76.15.227 strecopy

Synopsis: Copy a null-terminated string, return a pointer to the end.

Declaration: `function StrECopy(dest: PAnsiChar; source: PAnsiChar) : PAnsiChar`
`function strecopy(dest: PWideChar; source: PWideChar) : PWideChar`

Visibility: default

Description: Copies the Null-terminated string in `Source` to `Dest`, and returns a pointer to the end (i.e. the terminating Null-character) of the copied string.

Errors: No length checking is performed.

See also: `StrLCopy` (1761), `StrCopy` (1755)

Listing: `./examples/stringex/ex6.pp`

Program Example6;

Uses strings;

{ Program to demonstrate the StrECopy function. }

Const P : PChar = 'This is a PCHAR string.';

Var PP : PChar;

begin

PP:=StrAlloc (StrLen(P)+1);

If Longint(StrECopy(PP,P))-Longint(PP)<>StrLen(P) **then**

Writeln('Something is wrong here !')

else

Writeln ('PP= ',PP);

StrDispose(PP);

end.

76.15.228 strend

Synopsis: Return a pointer to the end of a null-terminated string.

Declaration: `function StrEnd(p: PAnsiChar) : PAnsiChar`
`function strend(p: PWideChar) : PWideChar`

Visibility: default

Description: Returns a pointer to the end of P. (i.e. to the terminating null-character.

Errors: None.

See also: StrLen ([1762](#))

Listing: ./examples/stringex/ex7.pp

Program Example6;

Uses strings;

{ Program to demonstrate the StrEnd function. }

Const P : PChar = 'This is a PCHAR string.';

begin

If Longint(StrEnd(P))-Longint(P)<>StrLen(P) **then**

Writeln('Something is wrong here !')

else

Writeln ('All is well..');

end.

76.15.229 StrFmt

Synopsis: Format a string with given arguments, store the result in a buffer.

Declaration: `function StrFmt(Buffer: PChar; Fmt: PChar; const args: Array of const)`
`: PChar`
`function StrFmt(Buffer: PChar; Fmt: PChar; const Args: Array of const;`
`const FormatSettings: TFormatSettings) : PChar`

Visibility: default

Description: StrFmt will format fmt with Args, as the Format ([1715](#)) function does, and it will store the result in Buffer. The function returns Buffer. Buffer should point to enough space to contain the whole result.

Errors: for a list of errors, see Format ([1715](#)).

See also: StrLFmt ([1762](#)), FmtStr ([1714](#)), Format ([1715](#)), FormatBuf ([1722](#))

Listing: ./examples/sysutex/ex80.pp

Program Example80;

{ This program demonstrates the StrFmt function }

Uses sysutils;

Var S : AnsiString;

Begin

SetLength(S,80);

Writeln (**StrFmt** (@S[1], 'For some nice examples of fomattng see %s.', ['Format']));

End.

76.15.230 stricomp

Synopsis: Compare 2 null-terminated strings, case insensitive.

Declaration: `function StrIComp(str1: PAnsiChar; str2: PAnsiChar) : SizeInt`
`function stricomp(str1: PWideChar; str2: PWideChar) : SizeInt`

Visibility: default

Description: Compares the null-terminated strings S1 and S2, ignoring case. The result is

- A negative Longint when S1<S2.
- 0 when S1=S2.
- A positive Longint when S1>S2.

Errors: None.

See also: StrLComp ([1760](#)), StrComp ([1755](#)), StrLComp ([1763](#))

Listing: ./examples/stringex/ex8.pp

Program Example8;

Uses strings;

{ Program to demonstrate the StrLComp function. }

Const P1 : PChar = 'This is the first string.';
P2 : PChar = 'This is the second string.';

Var L : Longint;

begin

Write ('P1 and P2 are ');

If StrComp (P1,P2)<>0 **then write** ('NOT ');

write ('equal. The first ');

L:=1;

While StrLComp(P1,P2,L)=0 **do inc** (L);

dec(L);

Writeln (L, 'characters are the same.');

end.

76.15.231 StringOf

Synopsis: Create a Unicode string from an array of bytes.

Declaration: `function StringOf(const Bytes: TBytes) : UnicodeString`

Visibility: default

Description: `StringOf` converts an array of bytes (`Bytes`) to a Unicode string. It interprets the bytes as a single-byte string, using the default codepage.

To create a string where the bytes are interpreted as wide chars, use `WideStringOf` ([1799](#)) instead.

Errors: None.

See also: `WideBytesOf` ([1796](#)), `WideStringOf` ([1799](#))

76.15.232 `StringReplace`

Synopsis: Replace occurrences of one substring with another in a string.

Declaration:

```
function StringReplace(const S: Ansistring;
                      const OldPattern: Ansistring;
                      const NewPattern: Ansistring;
                      Flags: TReplaceFlags; out aCount: Integer)
    : ansistring
function StringReplace(const S: Ansistring;
                      const OldPattern: Ansistring;
                      const NewPattern: Ansistring;
                      Flags: TReplaceFlags) : ansistring
```

Visibility: default

Description: `StringReplace` searches the string `S` for occurrences of the string `OldPattern` and, if it is found, replaces it with `NewPattern`. It returns the resulting string. The behaviour of `StringReplace` can be tuned with `Flags`, which is of type `TReplaceFlags` ([1637](#)). Standard behaviour is to replace only the first occurrence of `OldPattern`, and to search case sensitively.

Errors: None.

See also: `TReplaceFlags` ([1637](#)), `WideStringReplace` ([1799](#))

76.15.233 `StringToGUID`

Synopsis: Convert a string to a native `TGUID` type.

Declaration: `function StringToGUID(const S: string) : TGuid`

Visibility: default

Description: `StringToGUID` converts the string `S` to a valid GUID. The string `S` should be of the form

```
{XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}
```

Where each `X` is a hexadecimal digit. The dashes and braces are required.

Errors: In case `S` contains an invalid GUID representation, a `EConvertError` ([1812](#)) exception is raised.

See also: `Supports` ([1779](#)), `#rtl.system.TGUID` ([1583](#)), `GUIDToString` ([1734](#)), `IsEqualGuid` ([1739](#))

76.15.234 strlcat

Synopsis: Concatenate 2 null-terminated strings, with length boundary.

Declaration: `function StrLCat (dest: PAnsiChar; source: PAnsiChar; l: SizeInt)
 : PAnsiChar
 function strlcat (dest: PWideChar; source: PWideChar; l: SizeInt)
 : PWideChar`

Visibility: default

Description: Adds MaxLen characters from Source to Dest, and adds a terminating null-character. Returns Dest.

Errors: None.

See also: StrCat ([1754](#))

Listing: ./examples/stringex/ex12.pp

Program Example12;

Uses strings;

{ Program to demonstrate the StrLCat function. }

Const P1 : PChar = '1234567890';

Var P2 : PChar;

begin

 P2:=StrAlloc (StrLen(P1)*2+1);

 P2^:=#0; *{ Zero length }*

 StrCat (P2,P1);

 StrLCat (P2,P1,5);

 Writeln ('P2 = ',P2);

 StrDispose(P2)

end.

76.15.235 strlcomp

Synopsis: Compare limited number of characters of 2 null-terminated strings.

Declaration: `function StrLComp (str1: PAnsiChar; str2: PAnsiChar; l: SizeInt)
 : SizeInt
 function strlcomp (str1: PWideChar; str2: PWideChar; l: SizeInt)
 : SizeInt`

Visibility: default

Description: Compares maximum L characters of the null-terminated strings S1 and S2. The result is

- A negative Longint when S1<S2.
- 0 when S1=S2.
- A positive Longint when S1>S2.

Errors: None.

See also: [StrComp \(1755\)](#), [StrIComp \(1758\)](#), [StrLComp \(1763\)](#)

Listing: ./examples/stringex/ex8.pp

Program Example8;

Uses strings;

{ Program to demonstrate the StrLComp function. }

Const P1 : PChar = 'This is the first string.';
P2 : PChar = 'This is the second string.';

Var L : Longint;

begin

Write ('P1 and P2 are ');
If StrComp (P1,P2)<>0 **then write** ('NOT ');
write ('equal. The first ');
L:=1;
While StrLComp(P1,P2,L)=0 **do inc** (L);
dec(L);
Writeln (L,' characters are the same.');

end.

76.15.236 StrLCopy

Synopsis: Copy a null-terminated string, limited in length.

Declaration: function StrLCopy(dest: PAnsiChar; source: PAnsiChar; maxlen: SizeInt)
: PAnsiChar; Overload
function StrLCopy(Dest: PWideChar; Source: PWideChar; MaxLen: SizeInt)
: PWideChar; Overload

Visibility: default

Description: Copies MaxLen characters from Source to Dest, and makes Dest a null terminated string.

Errors: No length checking is performed.

See also: [StrCopy \(1755\)](#), [StrECopy \(1756\)](#)

Listing: ./examples/stringex/ex5.pp

Program Example5;

Uses strings;

{ Program to demonstrate the StrLCopy function. }

Const P : PChar = '123456789ABCDEF';

var PP : PChar;

begin

PP:=StrAlloc(11);
Writeln ('First 10 characters of P : ',StrLCopy (PP,P,10));
StrDispose(PP);

end.

76.15.237 StrLen

Synopsis: Length of a null-terminated string.

Declaration: `function StrLen(p: PAnsiChar) : SizeInt; Overload`
`function StrLen(p: PWideChar) : SizeInt; Overload`

Visibility: default

Description: Returns the length of the null-terminated string P. If P equals Nil then zero (0) is returned.

Errors: None.

See also: StrNew ([1764](#))

Listing: ./examples/stringex/ex1.pp

Program Example1;

Uses strings;

{ Program to demonstrate the StrLen function. }

Const P : PChar = 'This is a constant pchar string';

begin

Writeln ('P : ',p);

Writeln ('length(P) : ',StrLen(P));

end.

76.15.238 StrLFmt

Synopsis: Format a string with given arguments, but with limited length.

Declaration: `function StrLFmt(Buffer: PChar; Maxlen: Cardinal; Fmt: PChar;`
`const args: Array of const) : PChar`
`function StrLFmt(Buffer: PChar; Maxlen: Cardinal; Fmt: PChar;`
`const args: Array of const;`
`const FormatSettings: TFormatSettings) : PChar`

Visibility: default

Description: StrLFmt will format fmt with Args, as the Format ([1715](#)) function does, and it will store maximally Maxlen characters of the result in Buffer. The function returns Buffer. Buffer should point to enough space to contain MaxLen characters.

Errors: for a list of errors, see Format ([1715](#)).

See also: StrFmt ([1757](#)), FmtStr ([1714](#)), Format ([1715](#)), FormatBuf ([1722](#))

Listing: ./examples/sysutex/ex81.pp

Program Example80;

{ This program demonstrates the StrFmt function }

Uses sysutils;

Var S : AnsiString;

Begin

```
SetLength(S,80);
```

```
WriteLn (StrLFmt (@S[1],80,'For some nice examples of fomating see %s.',['Format']));
```

End.**76.15.239 strlicomp**

Synopsis: Compare limited number of characters in 2 null-terminated strings, ignoring case.

Declaration:

```
function StrLIComp(str1: PAnsiChar; str2: PAnsiChar; l: SizeInt)
    : SizeInt
function strlicomp(str1: PWideChar; str2: PWideChar; l: SizeInt)
    : SizeInt
```

Visibility: default

Description: Compares maximum L characters of the null-terminated strings S1 and S2, ignoring case. The result is

- A negative Longint when S1<S2.
- 0 when S1=S2.
- A positive Longint when S1>S2.

For an example, see StrIComp ([1758](#))

Errors: None.

See also: StrLComp ([1760](#)), StrComp ([1755](#)), StrIComp ([1758](#))

76.15.240 strlower

Synopsis: Convert null-terminated string to all-lowercase.

Declaration:

```
function StrLower(p: PAnsiChar) : PAnsiChar
function strlower(p: PWideChar) : PWideChar
```

Visibility: default

Description: Converts P to an all-lowercase string. Returns P.

Errors: None.

See also: StrUpper ([1779](#))

Listing: ./examples/stringex/ex14.pp

Program Example14;

Uses strings;

{ Program to demonstrate the StrLower and StrUpper functions. }

Const

```
P1 : PChar = 'THIS IS AN UPPERCASE PCHAR STRING';
P2 : PChar = 'this is a lowercase string';
```

```
begin
  P1:=StrNew(P1);
  P2:=strNew(P2);
  Writeln ('Uppercase : ',StrUpper(P2));
  StrLower(P1);
  Writeln ('Lowercase : ',P1);
  StrDispose(P1);
  StrDispose(P2);
end.
```

76.15.241 StrMove

Synopsis: Move a null-terminated string to new location.

```
Declaration: function StrMove(dest: PAnsiChar; source: PAnsiChar; l: SizeInt)
              : PAnsiChar; Overload
function StrMove(dest: PWideChar; source: PWideChar; l: SizeInt)
              : PWideChar; Overload
```

Visibility: default

Description: Copies `MaxLen` characters from `Source` to `Dest`. No terminating null-character is copied. Returns `Dest`

Errors: None.

See also: [StrLCopy \(1761\)](#), [StrCopy \(1755\)](#)

Listing: ./examples/stringex/ex10.pp

Program Example10;

Uses strings;

```
{ Program to demonstrate the StrMove function. }
```

```
Const P1 : PCHAR = 'This is a pchar string.';
```

```
Var P2 : Pchar;
```

```
begin
  P2:=StrAlloc(StrLen(P1)+1);
  StrMove (P2,P1,StrLen(P1)+1); { P2:=P1 }
  Writeln ('P2 = ',P2);
  StrDispose(P2);
end.
```

76.15.242 strnew

Synopsis: Allocate room for new null-terminated string.

```
Declaration: function StrNew(p: PAnsiChar) : PAnsiChar; Overload
             function strnew(p: PWideChar) : PWideChar; Overload
```

Visibility: default

Description: Copies `P` to the Heap, and returns a pointer to the copy.

Errors: Returns `Nil` if no memory was available for the copy.

See also: `StrCopy` ([1755](#)), `StrDispose` ([1756](#))

Listing: `./examples/stringex/ex16.pp`

Program `Example16`;

Uses `strings`;

{ Program to demonstrate the StrNew function. }

Const `P1 : PChar = 'This is a PChar string';`

var `P2 : PChar;`

begin

`P2:=StrNew (P1);`

If `P1=P2` **then**

`writeln ('This can't be happening...')`

else

`writeln ('P2 : ',P2);`

`StrDispose(P2);`

end.

76.15.243 StrNextChar

Synopsis: Returns a pointer to the location of the next empty character in a null-terminated string.

Declaration: `function StrNextChar(const Str: PAnsiChar) : PAnsiChar`

Visibility: default

Description: `StrNextChar` returns a pointer to the null-character that terminates the string `Str`

Errors: if `Str` is not properly terminated, an access violation may occur.

76.15.244 StrPas

Synopsis: Convert a null-terminated string to an ansistring.

Declaration: `function StrPas(Str: PAnsiChar) : string; Overload`
`function StrPas(Str: PWideChar) : UnicodeString; Overload`

Visibility: default

Description: Converts a null terminated string in `Str` to an `AnsiString`, and returns this string. This string is NOT truncated at 255 characters as is the system unit's version.

Errors: None.

See also: `StrPCopy` ([1766](#)), `StrPLCopy` ([1766](#))

76.15.245 StrPCopy

Synopsis: Copy an ansistring to a null-terminated string.

Declaration: `function StrPCopy(Dest: PAnsiChar; const Source: RawByteString)
: PAnsiChar; Overload
function StrPCopy(Dest: PWideChar; const Source: UnicodeString)
: PWideChar; Overload`

Visibility: default

Description: `StrPCopy` Converts the Ansistring in `Source` to a Null-terminated string, and copies it to `Dest`. `Dest` needs enough room to contain the string `Source`, i.e. `Length(Source)+1` bytes.

Errors: No checking is performed to see whether `Dest` points to enough memory to contain `Source`.

See also: `StrPLCopy` ([1766](#)), `StrPas` ([1765](#))

76.15.246 StrPLCopy

Synopsis: Copy a limited number of characters from an ansistring to a null-terminated string.

Declaration: `function StrPLCopy(Dest: PAnsiChar; const Source: RawByteString;
MaxLen: SizeUInt) : PAnsiChar; Overload
function StrPLCopy(Dest: PWideChar; const Source: UnicodeString;
MaxLen: SizeUInt) : PWideChar; Overload`

Visibility: default

Description: `StrPLCopy` Converts maximally `MaxLen` characters of the Ansistring in `Source` to a Null-terminated string, and copies it to `Dest`. `Dest` needs enough room to contain the characters.

Errors: No checking is performed to see whether `Dest` points to enough memory to contain `L` characters of `Source`.

See also: `StrPCopy` ([1766](#))

76.15.247 strpos

Synopsis: Find position of one null-terminated substring in another.

Declaration: `function StrPos(str1: PAnsiChar; str2: PAnsiChar) : PAnsiChar
function strpos(str1: PWideChar; str2: PWideChar) : PWideChar`

Visibility: default

Description: Returns a pointer to the first occurrence of `S2` in `S1`. If `S2` does not occur in `S1`, returns `Nil`.

Errors: None.

See also: `StrScan` ([1767](#)), `StrRScan` ([1767](#))

Listing: `./examples/stringex/ex15.pp`

Program Example15;

Uses strings;

{ Program to demonstrate the StrPos function. }

```

Const P : PChar = 'This is a PChar string.';
      S : Pchar = 'is';
begin
  Writeln ('Position of "is" in P : ',sizeint(StrPos(P,S))-sizeint(P));
end.

```

76.15.248 strscan

Synopsis: Find last occurrence of a character in a null-terminated string.

Declaration: `function StrRScan(p: PAnsiChar; c: AnsiChar) : PAnsiChar`
`function strscan(p: PWideChar; c: WideChar) : PWideChar`

Visibility: default

Description: Returns a pointer to the last occurrence of the character C in the null-terminated string P. If C does not occur, returns Nil.

For an example, see StrScan ([1767](#)).

Errors: None.

See also: StrScan ([1767](#)), StrPos ([1766](#))

76.15.249 StrScan

Synopsis: Find first occurrence of a character in a null-terminated string.

Declaration: `function StrScan(p: PAnsiChar; c: AnsiChar) : PAnsiChar; Overload`
`function StrScan(P: PWideChar; C: WideChar) : PWideChar; Overload`

Visibility: default

Description: Returns a pointer to the first occurrence of the character C in the null-terminated string P. If C does not occur, returns Nil.

Errors: None.

See also: StrRScan ([1767](#)), StrPos ([1766](#))

Listing: ./examples/stringex/ex13.pp

Program Example13;

Uses strings;

{ Program to demonstrate the StrScan and StrRScan functions. }

```

Const P : PChar = 'This is a PCHAR string.';
      S : Char = 's' ;

```

```

begin
  Writeln ('P, starting from first "s" : ',StrScan(P,s));
  Writeln ('P, starting from last "s" : ',StrRScan(P,s));
end.

```

76.15.250 StrToBool

Synopsis: Convert a string to a boolean value.

Declaration: `function StrToBool(const S: string) : Boolean`
`function StrToBool(const S: string;`
`const FormatSettings: TFormatSettings) : Boolean`

Visibility: default

Description: `StrToBool` will convert the string `S` to a boolean value. The string `S` can contain one of 'True', 'False' (case is ignored) or a numerical value. If it contains a numerical value, 0 is converted to False, all other values result in True. If the string `S` contains no valid boolean, then an `EConvertError` (1812) exception is raised.

Errors: On error, an `EConvertError` (1812) exception is raised.

See also: `BoolToStr` (1664)

76.15.251 StrToBoolDef

Synopsis: Convert string to boolean value, returning default in case of error.

Declaration: `function StrToBoolDef(const S: string; Default: Boolean) : Boolean`
`function StrToBoolDef(const S: string; Default: Boolean;`
`const FormatSettings: TFormatSettings) : Boolean`

Visibility: default

Description: `StrToBoolDef` tries to convert the string `S` to a boolean value, and returns the boolean value in case of success. In case `S` does not contain a valid boolean string, `Default` is returned.

See also: `StrToBool` (1768), `TryStrToBool` (1787)

76.15.252 StrToCurr

Synopsis: Convert a string to a currency value.

Declaration: `function StrToCurr(const S: string) : Currency`
`function StrToCurr(const S: string;`
`const FormatSettings: TFormatSettings) : Currency`

Visibility: default

Description: `StrToCurr` converts a string to a currency value and returns the value. The string should contain a valid currency amount, without currency symbol. If the conversion fails, an `EConvertError` (1812) exception is raised.

Errors: On error, an `EConvertError` (1812) exception is raised.

See also: `CurrToStr` (1673), `StrToCurrDef` (1769)

76.15.253 StrToCurrDef

Synopsis: Convert a string to a currency value, using a default value.

Declaration: `function StrToCurrDef(const S: string; Default: Currency) : Currency`
`function StrToCurrDef(const S: string; Default: Currency;`
`const FormatSettings: TFormatSettings) : Currency`

Visibility: default

Description: `StrToCurrDef` converts a string to a currency value and returns the value. The string should contain a valid currency amount, without currency symbol. If the conversion fails, the fallback `Default` value is returned.

Errors: On error, the `Default` value is returned.

See also: `CurrToStr` ([1673](#)), `StrToCurr` ([1768](#))

76.15.254 StrToDate

Synopsis: Convert a date string to a `TDateTime` value.

Declaration: `function StrToDate(const S: ShortString) : TDateTime`
`function StrToDate(const S: AnsiString) : TDateTime`
`function StrToDate(const S: ShortString; separator: AnsiChar)`
`: TDateTime`
`function StrToDate(const S: AnsiString; separator: AnsiChar) : TDateTime`
`function StrToDate(const S: string; FormatSettings: TFormatSettings)`
`: TDateTime`
`function StrToDate(const S: ShortString; const useformat: string;`
`separator: AnsiChar) : TDateTime`
`function StrToDate(const S: AnsiString; const useformat: string;`
`separator: AnsiChar) : TDateTime`
`function StrToDate(const S: PAnsiChar; Len: Integer;`
`const useformat: string; separator: AnsiChar)`
`: TDateTime`

Visibility: default

Description: `StrToDate` converts the string `S` to a `TDateTime` date value. The Date must consist of 1 to three digits, separated by the `DateSeparator` character. If two numbers are given, they are supposed to form the day and month of the current year. If only one number is given, it is supposed to represent the day of the current month. (This is *not* supported in Delphi)

The order of the digits (y/m/d, m/d/y, d/m/y) is determined from the `ShortDateFormat` variable.

Errors: On error (e.g. an invalid date or invalid character), an `EConvertError` ([1812](#)) exception is raised.

See also: `StrToTime` ([1776](#)), `DateToStr` ([1677](#)), `TimeToStr` ([1783](#)), `EConvertError` ([1812](#))

Listing: `./examples/sysutex/ex19.pp`

Program Example19;

{ This program demonstrates the StrToDate function }

Uses sysutils;

Procedure TestStr (S : String);

```

begin
  Writeln (S,': ',DateToStr(StrToDate(S)));
end;

Begin

  Writeln ('ShortDateFormat ',ShortDateFormat);
  TestStr(DateTimeToStr(Date));
  TestStr('05'+DateSeparator+'05'+DateSeparator+'1999');
  TestStr('5'+DateSeparator+'5');
  TestStr('5');
End.

```

76.15.255 StrToDateDef

Synopsis: Convert string to date, returning a default value.

Declaration: function StrToDateDef(const S: ShortString; const Defvalue: TDateTime) : TDateTime

```

function StrToDateDef(const S: ShortString; const Defvalue: TDateTime;
                      separator: AnsiChar) : TDateTime
function StrToDateDef(const S: AnsiString; const Defvalue: TDateTime)
                      : TDateTime
function StrToDateDef(const S: AnsiString; const Defvalue: TDateTime;
                      separator: AnsiChar) : TDateTime

```

Visibility: default

Description: StrToDateDef tries to convert the string S to a valid TDateTime date value, and returns DefValue if S does not contain a valid date indication.

Errors: None.

See also: StrToDate ([1769](#)), TryStrToDate ([1787](#)), StrToTimeDef ([1777](#))

76.15.256 StrToDateTime

Synopsis: Convert a date/time string to a TDateTime value.

Declaration: function StrToDateTime(const S: AnsiString) : TDateTime

```

function StrToDateTime(const s: ShortString;
                      const FormatSettings: TFormatSettings) : TDateTime
function StrToDateTime(const s: AnsiString;
                      const FormatSettings: TFormatSettings) : TDateTime

```

Visibility: default

Description: StrToDateTime converts the string S to a TDateTime date and time value. The date and time parts must be separated by a space.

For the date part, the same restrictions apply as for the StrToDate ([1769](#)) function: The Date must consist of 1 to three numbers, separated by the DateSeparator character. If two numbers are given, they are supposed to form the day and month of the current year. If only one number is given, it is supposed to represent the day of the current month. (This is *not* supported in Delphi)

The order of the 3 numbers (y/m/d, m/d/y, d/m/y) is determined from the ShortDateFormat variable.

Errors: On error (e.g. an invalid date or invalid character), an EConvertError (1812) exception is raised.

See also: StrToDate (1769), StrToTime (1776), DateTimeToStr (1674), EConvertError (1812)

Listing: ./examples/sysutex/ex20.pp

Program Example20;

{ This program demonstrates the StrToDateTime function }

Uses sysutils;

Procedure TestStr (S : String);

begin
 WriteLn (S, ' : ', DateTimeToStr(StrToDateTime(S)));
end;

Begin

WriteLn ('ShortDateFormat ', ShortDateFormat);
 TestStr(DateTimeToStr(**Now**));
 TestStr('05-05-1999 15:50');
 TestStr('5-5 13:30');
 TestStr('5 1:30PM');
End.

76.15.257 StrToDateTimeDef

Synopsis: Convert string to date/time, returning a default value.

Declaration: function StrToDateTimeDef(const S: ShortString;
 const Defvalue: TDateTime) : TDateTime
 function StrToDateTimeDef(const S: AnsiString;
 const Defvalue: TDateTime) : TDateTime
 function StrToDateTimeDef(const S: AnsiString;
 const Defvalue: TDateTime;
 const FormatSettings: TFormatSettings)
 : TDateTime

Visibility: default

Description: StrToDateTimeDef tries to convert the string S to a valid TDateTime date and time value, and returns DefValue if S does not contain a valid date-time indication.

Errors: None.

See also: StrToTimeDef (1777), StrToDateDef (1770), TryStrToDateTime (1788), StrToDateTime (1770)

76.15.258 StrToDWord

Synopsis: Convert string to DWord (cardinal).

Declaration: function StrToDWord(const s: string) : DWord

Visibility: default

Description: `StrToDWord` will convert the string `S` to a `DWord` value, and returns the value.

Errors: In case the string `S` is not a valid number, or is a value outside the `DWord` range, an `EConvertError` exception will be raised.

See also: `TryStrToDWord` (1788), `StrToDWordDef` (1772)

76.15.259 StrToDWordDef

Synopsis: Convert string to `DWord` (cardinal), using default.

Declaration: `function StrToDWordDef(const S: string; Default: DWord) : DWord`

Visibility: default

Description: `StrToDWordDef` will try to convert the string `S` to a `DWord` value. If the conversion was successful, it returns the value.

If the conversion failed, the fallback value in `Default` is returned.

See also: `TryStrToDWord` (1788), `StrToDWord` (1771)

76.15.260 StrToFloat

Synopsis: Convert a string to a floating-point value.

Declaration: `function StrToFloat(const S: string) : Extended`
`function StrToFloat(const S: string;`
`const FormatSettings: TFormatSettings) : Extended`

Visibility: default

Description: `StrToFloat` converts the string `S` to a floating point value. `S` should contain a valid string representation of a floating point value (either in decimal or scientific notation). The `thousandseparator` character may however not be used.

Up to and including version 2.2.2 of the compiler, if the string contains a decimal value, then the decimal separator character can either be a `'.'` or the value of the `DecimalSeparator` variable.

As of version 2.3.1, the string may contain only the `DecimalSeparator` character. The dot `('')` can no longer be used instead of the `DecimalSeparator`.

Errors: If the string `S` doesn't contain a valid floating point string, then an exception will be raised.

See also: `TextToFloat` (1780), `FloatToStr` (1709), `FormatFloat` (1724), `StrToInt` (1773)

Listing: `./examples/sysutex/ex90.pp`

Program Example90;

```
{ This program demonstrates the StrToFloat function }
{$mode objfpc}
{$h+ }
```

Uses SysUtils;

Const

```
NrValues = 5;
TestStr : Array[1..NrValues] of string =
    ('1','1','-0,2','1,2E-4','0','1E4');
```

```

Procedure Testit;

Var
  I : Integer;
  E : Extended;

begin
  Writeln('Using DecimalSeparator : ',DecimalSeparator);
  For I:=1 to NrValues do
    begin
      Writeln('Converting : ',TestStr[i]);
      Try
        E:=StrToFloat(TestStr[i]);
        Writeln('Converted value : ',E);
      except
        On E : Exception do
          Writeln('Exception when converting : ',E.Message);
      end;
    end;
  end;

Begin
  DecimalSeparator:=',';
  Testit;
  DecimalSeparator:= '.';
  Testit;
End.

```

76.15.261 StrToFloatDef

Synopsis: Convert a string to a float, with a default value.

Declaration: `function StrToFloatDef(const S: string; const Default: Extended) : Extended`
`function StrToFloatDef(const S: string; const Default: Extended; const FormatSettings: TFormatSettings) : Extended`

Visibility: default

Description: `StrToFloatDef` tries to convert the string `S` to a floating point value, and returns this value. If the conversion fails for some reason, the value `Default` is returned instead.

Errors: None. On error, the `Default` value is returned.

76.15.262 StrToInt

Synopsis: Convert a string to an integer value.

Declaration: `function StrToInt(const s: string) : LongInt`

Visibility: default

Description: `StrToInt` will convert the string `S` to a 32-bit signed integer. If the string contains invalid characters or has an invalid format, then an `EConvertError` ([1812](#)) is raised.

To be successfully converted, a string can contain a combination of `numerical` characters, possibly preceded by a minus sign (-). Spaces and tab characters are only allowed at the start of the string. Internally, `Val` (1570) is used, the rules of `Val` apply.

The string `S` can contain a number in decimal, hexadecimal, binary or octal format, as described in the language reference. For enumerated values, the string must be the name of the enumerated value. The name is searched case insensitively.

For hexadecimal values, the prefix `'0x'` or `'x'` (case insensitive) may be used as well in addition to the customary number notations described in the language reference.

Errors: In case of error, an `EConvertError` is raised.

See also: `Val` (1570), `IntToStr` (1738), `StrToUInt` (1777), `StrToInt64` (1774), `StrToIntDef` (1775), `EConvertError` (1812)

Listing: `./examples/sysutex/ex82.pp`

Program Example82;

{ \$mode objfpc }

{ This program demonstrates the StrToInt function }

Uses sysutils;

Begin

Writeln (**StrToInt**('1234'));

Writeln (**StrToInt**('-1234'));

Writeln (**StrToInt**('0'));

Try

Writeln (**StrToInt**('12345678901234567890'));

except

On E : `EConvertError` **do**

Writeln ('Invalid number encountered');

end;

End.

76.15.263 StrToInt64

Synopsis: Convert a string to an `Int64` value.

Declaration: `function StrToInt64(const s: string) : Int64`

Visibility: `default`

Description: `StrToInt64` converts the string `S` to a `Int64` value, and returns this value. The string can only contain numerical characters, and optionally a minus sign as the first character. Whitespace is not allowed.

Hexadecimal values (starting with the `$` character) are supported.

Errors: On error, a `EConvertError` (1812) exception is raised.

See also: `TryStrToInt64` (1789), `StrToInt64Def` (1775), `StrToInt` (1773), `TryStrToInt` (1789), `StrToIntDef` (1775)

76.15.264 StrToInt64Def

Synopsis: Convert a string to an Int64 value, with a default value.

Declaration: `function StrToInt64Def(const S: string; Default: Int64) : Int64`

Visibility: default

Description: `StrToInt64Def` tries to convert the string `S` to a `Int64` value, and returns this value. If the conversion fails for some reason, the value `Default` is returned instead.

Errors: None. On error, the `Default` value is returned.

See also: `StrToInt64` ([1774](#)), `TryStrToInt64` ([1789](#)), `StrToInt` ([1773](#)), `TryStrToInt` ([1789](#)), `StrToIntDef` ([1775](#))

76.15.265 StrToIntDef

Synopsis: Convert a string to an integer value, with a default value.

Declaration: `function StrToIntDef(const S: string; Default: LongInt) : LongInt`

Visibility: default

Description: `StrToIntDef` will convert a string to an integer. If the string contains invalid characters or has an invalid format, then `Default` is returned.

To be successfully converted, a string can contain a combination of numerical characters, possibly preceded by a minus sign (-). Spaces are not allowed.

Errors: None.

See also: `IntToStr` ([1738](#)), `StrToInt` ([1773](#))

Listing: `./examples/sysutex/ex83.pp`

Program Example82;

{ \$mode objfpc }

{ This program demonstrates the StrToInt function }

Uses sysutils;

Begin

Writeln (**StrToIntDef**('1234',0));

Writeln (**StrToIntDef**('-1234',0));

Writeln (**StrToIntDef**('0',0));

Try

Writeln (**StrToIntDef**('12345678901234567890',0));

except

On E : **EConvertError** **do**

Writeln ('Invalid number encountered');

end;

End.

76.15.266 StrToQWord

Synopsis: Convert a string to a QWord.

```
Declaration: function StrToQWord(const s: string) : QWord
```

Visibility: default

Description: TryStrToQWord converts the string S to a valid QWord (unsigned 64-bit) value, and returns the result.

Errors: If the string `S` does not contain a valid `QWord` value, a `EConvertError` (1812) exception is raised.

See also: [TryStrToQWord \(1790\)](#), [StrToQWordDef \(1776\)](#), [StrToInt64 \(1774\)](#), [StrToInt \(1773\)](#)

76.15.267 StrToQWordDef

Synopsis: Try to convert a string to a QWord, returning a default value in case of failure.

```
Declaration: function StrToOWordDef(const S: string; Default: OWord) : OWord
```

Visibility: default

Description: `StrToQWordDef` tries to convert the string `S` to a valid `QWord` (unsigned 64-bit) value, and returns the result. If the conversion fails, the function returns the value passed in `Def`.

See also: [StrToQWord \(1776\)](#), [TryStrToQWord \(1790\)](#), [StrToInt64Def \(1775\)](#), [StrToIntDef \(1775\)](#)

76.15.268 StrToTime

Synopsis: Convert a time string to a `TDateTime` value.

```

Declaration: function StrToTime(const S: Shortstring) : TDateTime
               function StrToTime(const S: Ansistring) : TDateTime
               function StrToTime(const S: ShortString; separator: AnsiChar)
                           : TDateTime
               function StrToTime(const S: AnsiString; separator: AnsiChar) : TDateTime
               function StrToTime(const S: string; FormatSettings: TFormatSettings)
                           : TDateTime
               function StrToTime(const S: PAnsiChar; Len: Integer;
                           separator: AnsiChar) : TDateTime

```

Visibility: default

Description: `StrToTime` converts the string `S` to a `TDateTime` time value. The time must consist of 1 to 4 digits, separated by the `TimeSeparator` character. If two numbers are given, they are supposed to form the hour and minutes.

Errors: On error (e.g. an invalid date or invalid character), an `EConvertError` ([1812](#)) exception is raised.

See also: [StrToDate \(1769\)](#), [StrToDateTime \(1770\)](#), [TimeToStr \(1783\)](#), [EConvertError \(1812\)](#)

Listing: ./examples/sysutex/ex21.pp

Program Example21;

```
{ This program demonstrates the StrToTime function }
```

Uses sysutils;

Procedure TestStr (S : String);

```
begin
  Writeln (S,': ',TimeToStr(StrToTime(S)));
end;
```

```
Begin
  teststr (TimeToStr(Time));
  teststr ('12:00');
  teststr ('15:30');
  teststr ('3:30PM');
End.
```

76.15.269 StrToTimeDef

Synopsis: Convert string to time, returning a default value.

Declaration: function StrToTimeDef(const S: ShortString; const Defvalue: TDateTime)
: TDateTime
function StrToTimeDef(const S: ShortString; const Defvalue: TDateTime;
separator: AnsiChar) : TDateTime
function StrToTimeDef(const S: AnsiString; const Defvalue: TDateTime)
: TDateTime
function StrToTimeDef(const S: AnsiString; const Defvalue: TDateTime;
separator: AnsiChar) : TDateTime

Visibility: default

Description: StrToTimeDef tries to convert the string S to a valid TDateTime time value, and returns DefValue if S does not contain a valid time indication.

Errors: None.

See also: StrToTime ([1776](#)), TryStrToTime ([1790](#)), StrToDateDef ([1770](#))

76.15.270 StrToUInt

Synopsis: Convert a string to unsigned integer value.

Declaration: function StrToUInt(const s: string) : Cardinal

Visibility: default

Description: StrToUInt will convert the string S to an unsigned 32-bit integer. If the string contains invalid characters or has an invalid format, then an EConvertError ([1812](#)) is raised.

To be successfully converted, a string can contain a combination of numerical characters. Spaces are not allowed.

The string S can contain a number in decimal, hexadecimal, binary or octal format, as described in the language reference.

For hexadecimal values, the prefix '0x' or 'x' (case insensitive) may be used as well.

Errors: In case of error, an EConvertError is raised.

See also: UIntToStr ([1791](#)), StrToInt ([1773](#)), StrToInt64 ([1774](#)), StrToUInt64 ([1778](#)), StrToIntDef ([1775](#)), EConvertError ([1812](#))

76.15.271 StrToUInt64

Synopsis: Convert a string to unsigned integer value.

Declaration: `function StrToUInt64(const s: string) : UInt64`

Visibility: default

Description: `StrToUInt64` will convert the string `S` to an unsigned 64-bit integer. If the string contains invalid characters or has an invalid format, then an `EConvertError` (1812) is raised.

To be successfully converted, a string can contain a combination of `numerical` characters, possibly preceded by a minus sign (-). Spaces are not allowed.

The string `S` can contain a number in decimal, hexadecimal, binary or octal format, as described in the language reference.

For hexadecimal values, the prefix '0x' or 'x' (case insensitive) may be used as well.

Errors: In case of error, an `EConvertError` is raised.

See also: `UIntToStr` (1791), `StrToInt` (1773), `StrToInt64` (1774), `StrToUInt64` (1778), `StrToIntDef` (1775), `EConvertError` (1812)

76.15.272 StrToUInt64Def

Synopsis: Convert a string to an unsigned 64-bit integer value, with a default value.

Declaration: `function StrToUInt64Def(const S: string; Default: UInt64) : UInt64`

Visibility: default

Description: `StrToIntDef` will convert a string to an unsigned 64-bit integer. If the string contains invalid characters or has an invalid format, then `Default` is returned.

To be successfully converted, a string can contain a combination of `numerical` characters. Spaces are not allowed.

Errors: None.

See also: `IntToStr` (1738), `StrToUInt64` (1778), `TryStrToUInt64` (1791), `StrToInt64` (1774)

76.15.273 StrToUIntDef

Synopsis: Convert a string to an unsigned integer value, with a default value.

Declaration: `function StrToUIntDef(const S: string; Default: Cardinal) : Cardinal`

Visibility: default

Description: `StrToIntDef` will convert a string to an integer. If the string contains invalid characters or has an invalid format, then `Default` is returned.

To be successfully converted, a string can contain a combination of `numerical` characters. Spaces are not allowed.

Errors: None.

See also: `IntToStr` (1738), `StrToUInt` (1777), `TryStrToUInt` (1790), `StrToInt` (1773)

76.15.274 strupper

Synopsis: Convert null-terminated string to all-uppercase.

Declaration: `function StrUpper(p: PAnsiChar) : PAnsiChar`
`function strupper(p: PWideChar) : PWideChar`

Visibility: default

Description: Converts P to an all-uppercase string. Returns P.

For an example, see `StrLower` ([1763](#))

Errors: None.

See also: `StrLower` ([1763](#))

76.15.275 Supports

Synopsis: Check whether a class or given interface supports an interface.

Declaration: `function Supports(const Instance: IInterface; const AClass: TClass;
out Obj) : Boolean; Overload`
`function Supports(const Instance: IInterface; const IID: TGuid;
out Intf) : Boolean; Overload`
`function Supports(const Instance: TObject; const IID: TGuid; out Intf)
: Boolean; Overload`
`function Supports(const Instance: TObject; const IID: Shortstring;
out Intf) : Boolean; Overload`
`function Supports(const Instance: IInterface; const AClass: TClass)
: Boolean; Overload`
`function Supports(const Instance: IInterface; const IID: TGuid)
: Boolean; Overload`
`function Supports(const Instance: TObject; const IID: TGuid) : Boolean
; Overload`
`function Supports(const Instance: TObject; const IID: Shortstring)
: Boolean; Overload`
`function Supports(const AClass: TClass; const IID: TGuid) : Boolean
; Overload`
`function Supports(const AClass: TClass; const IID: Shortstring)
: Boolean; Overload`

Visibility: default

Description: `Supports` checks whether `Instance` supports the interface identified by `IID`. It returns `True` if it is supported, `False`. Optionally, a pointer to the interface is returned to `Intf`.

Errors: None.

See also: `StringToGUID` ([1759](#))

76.15.276 SysErrorMessage

Synopsis: Format a system error message.

Declaration: `function SysErrorMessage(ErrorCode: Integer) : string`

Visibility: default

Description: `SysErrorMessage` returns a string that describes the operating system error code `ErrorCode`.

Errors: This routine may not be implemented on all platforms.

See also: `EOSError` ([1818](#))

76.15.277 SystemTimeToDateTime

Synopsis: Convert a system time to a `TDateTime` value.

Declaration: `function SystemTimeToDateTime(const SystemTime: TSystemTime) : TDateTime`

Visibility: default

Description: `SystemTimeToDateTime` converts a `TSystemTime` record to a `TDateTime` style date/time indication.

Errors: None.

See also: `DateTimeToSystemTime` ([1676](#))

Listing: `./examples/sysutex/ex22.pp`

Program Example22;

{ This program demonstrates the SystemTimeToDateTime function }

Uses sysutils;

Var ST : TSystemTime;

Begin

`DateTimeToSystemTime(Now,ST);`

With St **do**

begin

`Writeln ('Today is ',year,'/',month,'/',Day);`

`Writeln ('The time is ',Hour,':',minute,':',Second,':',MilliSecond);`

end;

`Writeln ('Converted : ',DateTimeToStr(SystemTimeToDateTime(ST)));`

End.

76.15.278 TextToFloat

Synopsis: Convert a buffer to a float value.

Declaration: `function TextToFloat(Buffer: PChar; out Value: Extended) : Boolean`
`function TextToFloat(Buffer: PChar; out Value: Extended;`
`const FormatSettings: TFormatSettings) : Boolean`
`function TextToFloat(Buffer: PChar; out Value; ValueType: TFloatValue)`
`: Boolean`
`function TextToFloat(Buffer: PChar; out Value; ValueType: TFloatValue;`
`const FormatSettings: TFormatSettings) : Boolean`

Visibility: default

Description: `TextToFloat` converts the string in `Buffer` to a floating point value. `Buffer` should contain a valid string representation of a floating point value (either in decimal or scientific notation).

If the buffer contains a decimal value, then the decimal separator character must be the value of the `DecimalSeparator` variable.

Remark Note that this behaviour has changed, earlier implementations also allowed the use of '.' in addition to the decimal separator character.

The function returns `True` if the conversion was successful.

Errors: If there is an invalid character in the buffer, then the function returns `False`

See also: `StrToFloat` ([1772](#)), `FloatToStr` ([1709](#)), `FormatFloat` ([1724](#))

Listing: `./examples/sysutex/ex91.pp`

Program Example91;

```
{ This program demonstrates the TextToFloat function }
{$mode objfpc}
{$h+ }
```

Uses SysUtils;

Const

```
NrValues = 5;
TestStr : Array[1..NrValues] of pchar =
    ('1,1','-0,2','1,2E-4','0','1E4');
```

Procedure Testit;

Var

```
I : Integer;
E : Extended;
```

begin

```
  Writeln('Using DecimalSeparator : ',DecimalSeparator);
```

```
  For I:=1 to NrValues do
```

```
    begin
```

```
      Writeln('Converting : ',TestStr[i]);
```

```
      If TextToFloat(TestStr[i],E) then
```

```
        Writeln('Converted value : ',E)
```

```
      else
```

```
        Writeln('Unable to convert value.');
```

```
      end;
```

```
end;
```

Begin

```
  DecimalSeparator:=',';
```

```
  Testit;
```

```
  DecimalSeparator:= '.';
```

```
  Testit;
```

```
End.
```

76.15.279 Time

Synopsis: Returns the current time.

Declaration: `function Time : TDateTime`

Visibility: default

Description: `Time` returns the current time in `TDateTime` format. The date part of the `TDateTimeValue` is set to zero.

Errors: None.

See also: `Now` ([1745](#)), `Date` ([1673](#))

Listing: `./examples/sysutex/ex23.pp`

Program Example23;

{ This program demonstrates the Time function }

Uses sysutils;

Begin

Writeln ('The time is : ', `TimeToStr(Time)`);

End.

76.15.280 TimeStampToDateTime

Synopsis: Convert a `TimeStamp` value to a `TDateTime` value.

Declaration: `function TimeStampToDateTime(const TimeStamp: TTimeStamp) : TDateTime`

Visibility: default

Description: `TimeStampToDateTime` converts `TimeStamp` to a `TDateTime` format variable. It is the inverse operation of `DateTimeToTimeStamp` ([1677](#)).

Errors: None.

See also: `DateTimeToTimeStamp` ([1677](#)), `TimeStampToMSecs` ([1783](#))

Listing: `./examples/sysutex/ex24.pp`

Program Example24;

{ This program demonstrates the TimeStampToDateTime function }

Uses sysutils;

Var TS : TTimeStamp;

DT : TDateTime;

Begin

TS:=`DateTimeToTimeStamp(Now)`;

With TS **do**

begin

Writeln ('Now is ', `time`, ' millisecond past midnight');

Writeln ('Today is ' ,`Date`, ' days past 1/1/0001');

end;

DT:=`TimeStampToDateTime(TS)`;

Writeln ('Together this is : ' ,`DateTimeToStr(DT)`);

End.

76.15.281 TimeStampToMSecs

Synopsis: Converts a timestamp to a number of milliseconds.

Declaration: `function TimeStampToMSecs(const TimeStamp: TTimeStamp) : Int64`

Visibility: default

Description: `TimeStampToMSecs` converts `TimeStamp` to the number of milliseconds since 1/1/0001.

Use `TTimeStamp` variables if you need to keep very precise track of time.

For an example, see `MSecsToTimeStamp` (1744).

Errors: None.

See also: `MSecsToTimeStamp` (1744), `TimeStampToDateTime` (1782)

76.15.282 TimeToStr

Synopsis: Convert a `TDateTime` time to a string using a predefined format.

Declaration: `function TimeToStr(Time: TDateTime) : string`
`function TimeToStr(Time: TDateTime;`
`const FormatSettings: TFormatSettings) : string`

Visibility: default

Description: `TimeToStr` converts the time in `Time` to a string. It uses the `LongTimeFormat` variable to see what formatting needs to be applied. It is therefor entirely equivalent to a `FormatDateTime('tt', Time)` call.

Note that on unix systems, the localization support must be enabled explicitly, see [Localization](#) (1610).

Errors: None.

Listing: `./examples/sysutex/ex25.pp`

Program Example25;

{ This program demonstrates the TimeToStr function }

Uses sysutils;

Begin

Writeln ('The current time is : ', `TimeToStr(Time)`);

End.

76.15.283 Trim

Synopsis: Trim whitespace from the ends of a string.

Declaration: `function Trim(const S: ansistring) : ansistring`
`function Trim(const S: WideString) : WideString`
`function Trim(const S: UnicodeString) : UnicodeString`

Visibility: default

Description: `Trim` strips blank characters (spaces and control characters) at the beginning and end of `S` and returns the resulting string. All characters with ordinal values less than or equal to 32 (a space) are stripped.

If the string contains only spaces, an empty string is returned.

Errors: None.

See also: `TrimLeft` ([1784](#)), `TrimRight` ([1785](#))

Listing: `./examples/sysutex/ex84.pp`

Program Example84;

{ This program demonstrates the Trim function }

Uses sysutils;
{ \$H+ }

Procedure Testit (S : **String**);

begin
 Writeln ("", Trim(S), "");
end;

Begin
 Testit (' ha ha what gets lost ? ');
 Testit (#10#13'haha ');
 Testit (' ');
End.

76.15.284 TrimLeft

Synopsis: Trim whitespace from the beginning of a string.

Declaration: `function TrimLeft(const S: ansistring) : ansistring`
`function TrimLeft(const S: WideString) : WideString`
`function TrimLeft(const S: UnicodeString) : UnicodeString`

Visibility: default

Description: `Trim` strips blank characters (spaces and control characters) at the beginning of `S` and returns the resulting string. All characters with ordinal values less than or equal to 32 (a space) are stripped.

If the string contains only spaces, an empty string is returned.

Errors: None.

See also: `Trim` ([1783](#)), `TrimRight` ([1785](#))

Listing: `./examples/sysutex/ex85.pp`

Program Example85;

{ This program demonstrates the TrimLeft function }

Uses sysutils;
{ \$H+ }

Procedure Testit (S : **String**);

begin

Writeln ('', TrimLeft(S), '');

end;

Begin

 Testit (' ha ha what gets lost ? ');

 Testit (#10#13'haha ');

 Testit (' ');

End.

76.15.285 TrimRight

Synopsis: Trim whitespace from the end of a string.

Declaration: `function TrimRight(const S: ansistring) : ansistring`
 `function TrimRight(const S: WideString) : WideString`
 `function TrimRight(const S: UnicodeString) : UnicodeString`

Visibility: default

Description: `Trim` strips blank characters (spaces and control characters) at the end of `S` and returns the resulting string. All characters with ordinal values less than or equal to 32 (a space) are stripped.

If the string contains only spaces, an empty string is returned.

Errors: None.

See also: `Trim` ([1783](#)), `TrimLeft` ([1784](#))

Listing: ./examples/sysutex/ex86.pp

Program Example86;

{ This program demonstrates the TrimRight function }

Uses sysutils;

{ \$H+ }

Procedure Testit (S : **String**);

begin

Writeln ('', TrimRight(S), '');

end;

Begin

 Testit (' ha ha what gets lost ? ');

 Testit (#10#13'haha ');

 Testit (' ');

End.

76.15.286 TryEncodeDate

Synopsis: Try to encode a date, and indicate success.

Declaration: `function TryEncodeDate(Year: Word; Month: Word; Day: Word;
out Date: TDateTime) : Boolean`

Visibility: default

Description: `TryEncodeDate` will check the validity of the `Year`, `Month` and `Day` arguments, and if they are all valid, then they will be encoded as a `TDateTime` value and returned in `Date`. The function will return `True` in this case. If an invalid argument is passed, then `False` will be returned.

Errors: None. If an error occurs during the encoding, `False` is returned.

See also: `EncodeDate` ([1683](#)), `DecodeDateFully` ([1679](#)), `DecodeDate` ([1678](#)), `TryEncodeTime` ([1786](#))

76.15.287 TryEncodeTime

Synopsis: Try to encode a time, and indicate success.

Declaration: `function TryEncodeTime(Hour: Word; Min: Word; Sec: Word; MSec: Word;
out Time: TDateTime) : Boolean`

Visibility: default

Description: `TryEncodeTime` will check the validity of the `Hour`, `Min`, `Sec` and `MSec` arguments, and will encode them in a `TDateTime` value which is returned in `Time`. If the arguments are valid, then `True` is returned, otherwise `False` is returned.

Errors: None. If an error occurs during the encoding, `False` is returned.

See also: `EncodeTime` ([1684](#)), `DecodeTime` ([1679](#)), `TryEncodeDate` ([1785](#))

76.15.288 TryFloatToCurr

Synopsis: Try to convert a float value to a currency value and report on success.

Declaration: `function TryFloatToCurr(const Value: Extended; var AResult: Currency)
: Boolean`

Visibility: default

Description: `TryFloatToCurr` tries convert the `Value` floating point value to a `Currency` value. If successful, the function returns `True` and the resulting currency value is returned in `AResult`. It checks whether `Value` is in the valid range of currencies (determined by `MinCurrency` ([1621](#)) and `MaxCurrency` ([1621](#))). If not, `False` is returned.

Errors: If `Value` is out of range, `False` is returned.

See also: `FloatToCurr` ([1708](#)), `MinCurrency` ([1621](#)), `MaxCurrency` ([1621](#))

76.15.289 TryStringToGUID

Synopsis: Try to transform a string to a GUID.

Declaration: `function TryStringToGUID(const S: string; out Guid: TGUID) : Boolean`

Visibility: default

Description: `TryStringToGUID` tries to convert the string `S` to a `TGUID` value, returned in `GUID`. It returns `True` if the conversion succeeds, and `False` if the string `S` does not contain a valid `GUID` notation. The string `S` must be 38 characters long, must start with `{` and end on `}`, and contain a valid `GUID` string (hex number grouped using 8-4-4-4-12 digits).

Errors: In case `S` does not contain a valid `GUID` number, `False` is returned.

See also: `StringToGUID` (1759)

76.15.290 TryStrToBool

Synopsis: Try to convert a string to a boolean value.

Declaration:

```
function TryStrToBool(const S: string; out Value: Boolean) : Boolean
function TryStrToBool(const S: string; out Value: Boolean;
                      const FormatSettings: TFormatSettings) : Boolean
```

Visibility: default

Description: `TryStrToBool` tries to convert the string `S` to a boolean value, and returns this value in `Value`. In this case, the function returns `True`. If `S` does not contain a valid boolean string, the function returns `False`, and the contents of `Value` is undetermined.

Valid boolean string constants are in the `FalseBoolStrs` (1643) (for `False` values) and `TrueBoolStrs` (1646) (for `True` values) variables.

See also: `StrToBool` (1768), `StrToBoolDef` (1768)

76.15.291 TryStrToCurr

Synopsis: Try to convert a string to a currency.

Declaration:

```
function TryStrToCurr(const S: string; out Value: Currency) : Boolean
function TryStrToCurr(const S: string; out Value: Currency;
                      const FormatSettings: TFormatSettings) : Boolean
```

Visibility: default

Description: `TryStrToCurr` converts the string `S` to a currency value and returns the value in `Value`. The function returns `True` if it was successful, `False` if not. This is contrary to `StrToCurr` (1768), which raises an exception when the conversion fails.

The function takes into account locale information.

See also: `StrToCurr` (1768), `TextToFloat` (1780)

76.15.292 TryStrToDate

Synopsis: Try to convert a string with a date indication to a `TDateTime` value.

Declaration:

```
function TryStrToDate(const S: ShortString; out Value: TDateTime)
                      : Boolean
function TryStrToDate(const S: AnsiString; out Value: TDateTime)
                      : Boolean
function TryStrToDate(const S: ShortString; out Value: TDateTime;
                      separator: AnsiChar) : Boolean
function TryStrToDate(const S: AnsiString; out Value: TDateTime;
```

```

        separator: AnsiChar) : Boolean
function TryStrToDate(const S: ShortString; out Value: TDateTime;
    const useformat: string; separator: AnsiChar)
    : Boolean
function TryStrToDate(const S: AnsiString; out Value: TDateTime;
    const useformat: string; separator: AnsiChar)
    : Boolean
function TryStrToDate(const S: string; out Value: TDateTime;
    const FormatSettings: TFormatSettings) : Boolean

```

Visibility: default

Description: TryStrToDate tries to convert the string *S* to a TDateTime date value, and stores the date in *Value*. The Date must consist of 1 to three digits, separated by the DateSeparator character. If two numbers are given, they are supposed to form the day and month of the current year. If only one number is given, it is supposed to represent the day of the current month. (This is *not* supported in Delphi)

The order of the digits (y/m/d, m/d/y, d/m/y) is determined from the ShortDateFormat variable.

The function returns True if the string contained a valid date indication, False otherwise.

See also: StrToDate (1769), StrToTime (1776), TryStrToTime (1790), TryStrToDateTime (1788), DateToStr (1677), TimeToStr (1783)

76.15.293 TryStrToDateTime

Synopsis: Try to convert a string with date/time indication to a TDateTime value.

```

Declaration: function TryStrToDateTime(const S: ShortString; out Value: TDateTime)
    : Boolean
function TryStrToDateTime(const S: AnsiString; out Value: TDateTime)
    : Boolean
function TryStrToDateTime(const S: string; out Value: TDateTime;
    const FormatSettings: TFormatSettings)
    : Boolean

```

Visibility: default

Description: TryStrToDateTime tries to convert the string *S* to a TDateTime date and time value, and stores the result in *Value*. The date must consist of 1 to three digits, separated by the DateSeparator character. If two numbers are given, they are supposed to form the day and month of the current year. If only one number is given, it is supposed to represent the day of the current month (This is *not* supported in Delphi). The time must consist of 1 to 4 digits, separated by the TimeSeparator character. If two numbers are given, they are supposed to form the hour and minutes.

The function returns True if the string contained a valid date and time indication, False otherwise.

See also: TryStrToDate (1787), TryStrToTime (1790), StrToDateTime (1770), StrToTime (1776), DateToStr (1677), TimeToStr (1783)

76.15.294 TryStrToDWord

Synopsis: Try to convert a string to DWord (cardinal).

```

Declaration: function TryStrToDWord(const s: string; out D: DWord) : Boolean

```

Visibility: default

Description: `TryStrToDWord` will try to convert the string `S` to a `DWord` value. It returns `True` if the conversion was successful, and in that case returns the value in `D`.

If the conversion failed or the value was outside of the valid range for `DWord` values, `False` is returned.

See also: `StrToDWord` ([1771](#)), `StrToDWordDef` ([1772](#))

76.15.295 TryStrToFloat

Synopsis: Try to convert a string to a float.

Declaration:

```
function TryStrToFloat(const S: string; out Value: Single) : Boolean
function TryStrToFloat(const S: string; out Value: Single;
                        const FormatSettings: TFormatSettings) : Boolean
function TryStrToFloat(const S: string; out Value: Double) : Boolean
function TryStrToFloat(const S: string; out Value: Double;
                        const FormatSettings: TFormatSettings) : Boolean
function TryStrToFloat(const S: string; out Value: Extended) : Boolean
function TryStrToFloat(const S: string; out Value: Extended;
                        const FormatSettings: TFormatSettings) : Boolean
```

Visibility: default

Description: `TryStrToFloat` tries to convert the string `S` to a floating point value, and stores the result in `Value`. It returns `True` if the operation was successful, and `False` if it failed. This operation takes into account the system settings for floating point representations.

Errors: On error, `False` is returned.

See also: `StrToFloat` ([1772](#))

76.15.296 TryStrToInt

Synopsis: Try to convert a string to an integer, and report on success.

Declaration: `function TryStrToInt(const s: string; out i: LongInt) : Boolean`

Visibility: default

Description: `TryStrToInt` tries to convert the string `S` to an integer, and returns `True` if this was successful. In that case the converted integer is returned in `I`. If the conversion failed, (an invalid string, or the value is out of range) then `False` is returned.

Errors: None. On error, `False` is returned.

See also: `StrToInt` ([1773](#)), `TryStrToUInt` ([1790](#)), `TryStrToInt64` ([1789](#)), `StrToIntDef` ([1775](#)), `StrToInt64` ([1774](#)), `StrToInt64Def` ([1775](#))

76.15.297 TryStrToInt64

Synopsis: Try to convert a string to an int64 value, and report on success.

Declaration: `function TryStrToInt64(const s: string; out i: Int64) : Boolean`

Visibility: default

Description: `TryStrToInt64` tries to convert the string `S` to a `Int64` value, and returns this value in `I` if successful. If the conversion was successful, the function result is `True`, or `False` otherwise. The string can only contain numerical characters, and optionally a minus sign as the first character. Whitespace is not allowed.

Hexadecimal values (starting with the `$` character) are supported.

Errors: None. On error, `False` is returned.

See also: `StrToInt64` ([1774](#)), `StrToInt64Def` ([1775](#)), `StrToInt` ([1773](#)), `TryStrToInt` ([1789](#)), `StrToIntDef` ([1775](#))

76.15.298 TryStrToQWord

Synopsis: Try to convert a string to a `QWord` value, and report on success.

Declaration: `function TryStrToQWord(const s: string; out Q: QWord) : Boolean`

Visibility: default

Description: `TryStrToQWord` tries to convert the string `S` to a valid `QWord` (unsigned 64-bit) value, and stores the result in `I`. If the conversion fails, the function returns `False`, else it returns `True`.

See also: `StrToQWord` ([1776](#)), `StrToQWordDef` ([1776](#)), `TryStrToInt64` ([1789](#)), `TryStrToInt` ([1789](#))

76.15.299 TryStrToTime

Synopsis: Try to convert a string with a time indication to a `TDateTime` value.

Declaration: `function TryStrToTime(const S: ShortString; out Value: TDateTime) : Boolean`
`function TryStrToTime(const S: AnsiString; out Value: TDateTime) : Boolean`
`function TryStrToTime(const S: ShortString; out Value: TDateTime; separator: AnsiChar) : Boolean`
`function TryStrToTime(const S: AnsiString; out Value: TDateTime; separator: AnsiChar) : Boolean`
`function TryStrToTime(const S: string; out Value: TDateTime; const FormatSettings: TFormatSettings) : Boolean`

Visibility: default

Description: `TryStrToTime` tries to convert the string `S` to a `TDateTime` time value, and stores the result in `Value`. The time must consist of 1 to 4 digits, separated by the `TimeSeparator` character. If two numbers are given, they are supposed to form the hour and minutes.

The function returns `True` if the string contained a valid time indication, `False` otherwise.

See also: `TryStrToDate` ([1787](#)), `TryStrToDateTime` ([1788](#)), `StrToDate` ([1769](#)), `StrToTime` ([1776](#)), `DateToStr` ([1677](#)), `TimeToStr` ([1783](#))

76.15.300 TryStrToUInt

Synopsis: Try to convert a string to an unsigned integer, and report on success.

Declaration: `function TryStrToUInt(const s: string; out C: Cardinal) : Boolean`

Visibility: default

Description: `TryStrToUInt` tries to convert the string `S` to an unsigned 32-bit integer (a `Cardinal`), and returns `True` if this was successful. In that case the converted unsigned integer is returned in `C`. If the conversion failed, (an invalid string, or the value is out of range) then `False` is returned.

Errors: None. On error, `False` is returned.

See also: `StrToInt` ([1773](#)), `TryStrToInt` ([1789](#)), `TryStrToInt64` ([1789](#)), `StrToIntDef` ([1775](#)), `StrToInt64` ([1774](#)), `StrToInt64Def` ([1775](#))

76.15.301 TryStrToUInt64

Synopsis: Try to convert a string to an unsigned 64-bit integer, and report on success.

Declaration: `function TryStrToUInt64(const s: string; out u: UInt64) : Boolean`

Visibility: default

Description: `TryStrToUInt` tries to convert the string `S` to an unsigned 64-bit integer (a `Cardinal`), and returns `True` if this was successful. In that case the converted unsigned integer is returned in `U`. If the conversion failed, (an invalid string, or the value is out of range) then `False` is returned.

Errors: None. On error, `False` is returned.

See also: `StrToInt` ([1773](#)), `TryStrToInt` ([1789](#)), `TryStrToInt64` ([1789](#)), `StrToIntDef` ([1775](#)), `StrToInt64` ([1774](#)), `StrToInt64Def` ([1775](#))

76.15.302 UIntToStr

Synopsis: Unsigned integer to String.

Declaration: `function UIntToStr(Value: QWord) : string`
`function UIntToStr(Value: Cardinal) : string`

Visibility: default

Description: `UIntToStr` is the unsigned counterpart to `IntToStr` ([1738](#)): it converts the unsigned integer `Value` to its string representation. The resulting string has only as much characters as needed to represent the value.

Errors: None

See also: `IntToStr` ([1738](#))

76.15.303 UnhookSignal

Synopsis: `UnHook` a specified signal.

Declaration: `procedure UnhookSignal(RtlSigNum: Integer; OnlyIfHooked: Boolean)`

Visibility: default

Description: `UnHookSignal` de-installs the RTL default signal handler for signal `RtlSigNum`. If `OnlyIfHooked` is `True` then `UnhookSignal` will first check if the signal was hooked by the RTL routines, and has not been overridden since.

76.15.304 UnicodeCompareStr

Synopsis: Compare 2 Unicode strings.

Declaration: `function UnicodeCompareStr(const s1: UnicodeString;
const s2: UnicodeString) : PtrInt`

Visibility: default

Description: `UnicodeCompareStr` compares 2 Unicode strings `S1` and `S2` in a case sensitive manner. The result of the function is

`< 0` If `S1<S2`

`0` If `S1=S2`

`> 0` If `S1>S2`

This function relies on a widestring manager to perform the actual comparison, as it will take into account various equivalent code points: it is not a simple byte-by-byte comparison.

See also: `UnicodeCompareText` ([1792](#)), `CompareStr` ([1668](#)), `CompareText` ([1669](#)), `UnicodeSameStr` ([1794](#))

76.15.305 UnicodeCompareText

Synopsis: Compare 2 strings case insensitively.

Declaration: `function UnicodeCompareText(const s1: UnicodeString;
const s2: UnicodeString) : PtrInt`

Visibility: default

Description: `UnicodeCompareText` compares 2 Unicode strings `S1` and `S2`, ignoring case. The result of the function is

`< 0` If `S1<S2`

`0` If `S1=S2`

`> 0` If `S1>S2`

This function relies on a widestring manager to perform the actual comparison, as it will take into account various equivalent code points: it is not a simple byte-by-byte comparison.

See also: `UnicodeCompareText` ([1792](#)), `CompareStr` ([1668](#)), `CompareText` ([1669](#)), `UnicodeSameStr` ([1794](#))

76.15.306 UnicodeFmtStr

Synopsis: Format a string with given arguments, procedural version.

Declaration: `procedure UnicodeFmtStr(var Res: UnicodeString;
const Fmt: UnicodeString;
const args: Array of const)
procedure UnicodeFmtStr(var Res: UnicodeString;
const Fmt: UnicodeString;
const args: Array of const;
const FormatSettings: TFormatSettings)`

Visibility: default

Description: `UnicodeFmtStr` calls `UnicodeFormat` (1793) with `Fmt` and `Args` as arguments, and stores the result in `Res`. For more information on how the resulting string is composed, see `UnicodeFormat` (1793).

Errors: In case of error, an `EConvertError` (1812) exception is raised.

See also: `UnicodeFormat` (1793), `UnicodeFormatBuf` (1793)

76.15.307 UnicodeFormat

Synopsis: Format Unicode string.

Declaration:

```
function UnicodeFormat(const Fmt: UnicodeString;
                      const Args: Array of const) : UnicodeString
function UnicodeFormat(const Fmt: UnicodeString;
                      const Args: Array of const;
                      const FormatSettings: TFormatSettings)
                      : UnicodeString
```

Visibility: default

Description: `UnicodeFormat` is the Unicode equivalent of `Format` (1715). It follows the same rules and uses the same formatting strings.

Errors: In case of error, an `EConvertError` (1812) exception is raised.

See also: `Format` (1715), `UnicodeFormatBuf` (1793)

76.15.308 UnicodeFormatBuf

Synopsis: Format a Unicode string with given arguments and store the result in a unicodebuffer.

Declaration:

```
function UnicodeFormatBuf(var Buffer; BufLen: Cardinal; const Fmt;
                        fmtLen: Cardinal; const Args: Array of const)
                        : Cardinal
function UnicodeFormatBuf(var Buffer; BufLen: Cardinal; const Fmt;
                        fmtLen: Cardinal; const Args: Array of const;
                        const FormatSettings: TFormatSettings)
                        : Cardinal
```

Visibility: default

Description: `UnicodeFormatBuf` is the Unicode equivalent of `FormatBuf` (1722). It follows the same rules and uses the same formatting st

Errors: In case of error, an `EConvertError` (1812) exception is raised.

See also: `FormatBuf` (1722), `UnicodeFormat` (1793)

76.15.309 UnicodeLowerCase

Synopsis: Return lowercase version of a string.

Declaration:

```
function UnicodeLowerCase(const s: UnicodeString) : UnicodeString
```

Visibility: default

Description: `UnicodeLowerCase` returns an all-lowercase version of the unicodestring `S`. It relies on the Unicode manager to do so.

See also: `UnicodeUpperCase` ([1795](#)), `UpperCase` ([1795](#)), `LowerCase` ([1743](#))

76.15.310 UnicodeSameStr

Synopsis: Check whether 2 strings are equal.

Declaration:

```
function UnicodeSameStr(const s1: UnicodeString;
                        const s2: UnicodeString) : Boolean
```

Visibility: default

Description: `UnicodeSameStr` checks whether the Unicode strings `S1` and `S2` are equal, case sensitively. The function returns `True` if the strings are equal, `False` if they are not. This function relies on a widestring manager to perform the actual comparison, as it will take into account various equivalent code points: it is not a simple byte-by-byte comparison.

See also: `UnicodeCompareText` ([1792](#)), `CompareStr` ([1668](#)), `CompareText` ([1669](#)), `UnicodeSameText` ([1794](#)), `SameStr` ([1750](#))

76.15.311 UnicodeSameText

Synopsis: Check whether 2 strings are the same, ignoring case.

Declaration:

```
function UnicodeSameText(const s1: UnicodeString;
                        const s2: UnicodeString) : Boolean
```

Visibility: default

Description: `UnicodeSameText` checks whether the Unicode strings `S1` and `S2` are equal, ignoring case. The function returns `True` if the strings are equal, `False` if they are not. This function relies on a widestring manager to perform the actual comparison, as it will take into account various equivalent code points: it is not a simple byte-by-byte comparison.

See also: `UnicodeCompareText` ([1792](#)), `CompareStr` ([1668](#)), `CompareText` ([1669](#)), `UnicodeSameStr` ([1794](#)), `SameText` ([1750](#))

76.15.312 UnicodeStringReplace

Synopsis: Replace one occurrence of a string with another.

Declaration:

```
function UnicodeStringReplace(const S: UnicodeString;
                             const OldPattern: UnicodeString;
                             const NewPattern: UnicodeString;
                             Flags: TReplaceFlags) : UnicodeString

function UnicodeStringReplace(const S: UnicodeString;
                             const OldPattern: UnicodeString;
                             const NewPattern: UnicodeString;
                             Flags: TReplaceFlags; out aCount: Integer)
                             : UnicodeString
```

Visibility: default

Description: `UnicodeStringReplace` is the Unicode version of `StringReplace` ([1759](#)); it follows the same rules and has the same behaviour, but operates on Unicode strings instead of ansistrings.

See also: `StringReplace` ([1759](#))

76.15.313 UnicodeUpperCase

Synopsis: Return uppercase version of a string.

Declaration: `function UnicodeUpperCase(const s: UnicodeString) : UnicodeString`

Visibility: default

Description: `UnicodeUpperCase` returns an all-uppercase version of the `UnicodeString` S. It relies on the Unicode manager to do so.

See also: `UnicodeLowerCase` ([1793](#)), `LowerCase` ([1743](#)), `UpperCase` ([1795](#))

76.15.314 UniversalTimeToLocal

Declaration: `function UniversalTimeToLocal(UT: TDateTime) : TDateTime`
`function UniversalTimeToLocal(UT: TDateTime; TZOffset: Integer)`
`: TDateTime`

Visibility: default

76.15.315 UniversalToFileDate

Declaration: `function UniversalToFileDate(DateTime: TDateTime) : Int64`

Visibility: default

76.15.316 UpperCase

Synopsis: Return an uppercase version of a string.

Declaration: `function UpperCase(const s: ansistring) : ansistring; Overload`
`function UpperCase(const s: ansistring; LocaleOptions: TLocaleOptions)`
`: ansistring; Overload`
`function UpperCase(const s: UnicodeString) : UnicodeString; Overload`

Visibility: default

Description: `UpperCase` returns the uppercase equivalent of S. Ansi characters are not taken into account, only ASCII codes below 127 are converted. It is completely equivalent to the `UpCase` function of the system unit, and is provided for compatibility only.

Errors: None.

See also: `AnsiLowerCase` ([1653](#)), `LowerCase` ([1743](#)), `AnsiUpperCase` ([1661](#))

Listing: `./examples/sysutex/ex87.pp`

Program Example87;

{ This program demonstrates the UpperCase function }

Uses sysutils;

Begin

Writeln (`UpperCase('this will come OUT ALL uPpErCaSe !')`);

End.

76.15.317 VendorName

Synopsis: Return Application vendor Name.

Declaration: `function VendorName : string`

Visibility: default

Description: `VendorName` returns the application vendor name. In order to set the application vendor name, the `OnGetVendorName` (1645) event must be set, and an appropriate return value must be returned. The Vendor name is used in `GetAppConfigDir` (1726) and `GetAppConfigFile` (1726) to determine the configuration directory.

Errors: If `OnGetVendorName` (1645) is not set, an empty string is returned.

See also: `OnGetVendorName` (1645), `GetAppConfigDir` (1726), `GetAppConfigFile` (1726)

76.15.318 WideBytesOf

Synopsis: Returns the contents of a widestring as an array of bytes.

Declaration: `function WideBytesOf(const Value: UnicodeString) : TBytes`

Visibility: default

Description: `WideBytesOf` returns the contents of the widestring `Value` as an array of bytes. The array will have as length twice the length of the wide string, as each wide character contains 2 bytes.

See also: `StringOf` (1758)

76.15.319 WideCompareStr

Synopsis: Compare two widestrings (case sensitive).

Declaration: `function WideCompareStr(const s1: WideString; const s2: WideString)
: PtrInt`

Visibility: default

Description: `WideCompareStr` compares two widestrings and returns the following result:

< 0 if `S1 < S2`.

0 if `S1 = S2`.

> 0 if `S1 > S2`.

The comparison takes into account wide characters, i.e. it takes care of strange accented characters. Contrary to `WideCompareText` (1797), the comparison is case sensitive.

Errors: None.

See also: `WideCompareText` (1797), `WideSameStr` (1798), `WideSameText` (1798)

76.15.320 WideCompareText

Synopsis: Compare two widestrings (ignoring case).

Declaration: `function WideCompareText(const s1: WideString; const s2: WideString)
: PtrInt`

Visibility: default

Description: `WideCompareStr` compares two widestrings and returns the following result:

< 0 if `S1`<`S2`.

0 if `S1`=`S2`.

> 0 if `S1`>`S2`.

The comparison takes into account wide characters, i.e. it takes care of strange accented characters. Contrary to `WideCompareStr` (1796), the comparison is case insensitive.

Errors: None.

See also: `WideCompareStr` (1796), `WideSameStr` (1798), `WideSameText` (1798)

76.15.321 WideFmtStr

Synopsis: Widestring format.

Declaration: `procedure WideFmtStr(var Res: WideString; const Fmt: WideString;
const args: Array of const)
procedure WideFmtStr(var Res: WideString; const Fmt: WideString;
const args: Array of const;
const FormatSettings: TFormatSettings)`

Visibility: default

Description: `WideFmtStr` formats `Args` according to the format string in `Fmt` and returns the resulting string in `Res`.

See also: `WideFormat` (1797), `WideFormatBuf` (1798), `Format` (1715)

76.15.322 WideFormat

Synopsis: Format a wide string.

Declaration: `function WideFormat(const Fmt: WideString; const Args: Array of const)
: WideString
function WideFormat(const Fmt: WideString; const Args: Array of const;
const FormatSettings: TFormatSettings) : WideString`

Visibility: default

Description: `WideFormat` does the same as `Format` (1715) but accepts as a formatting string a `WideString`. The resulting string is also a `WideString`.

For more information about the used formatting characters, see the `Format` (1715) string.

See also: `Format` (1715)

76.15.323 WideFormatBuf

Synopsis: Format widestring in a buffer.

Declaration:

```
function WideFormatBuf(var Buffer; BufLen: Cardinal; const Fmt;
                        fmtLen: Cardinal; const Args: Array of const)
                        : Cardinal
function WideFormatBuf(var Buffer; BufLen: Cardinal; const Fmt;
                        fmtLen: Cardinal; const Args: Array of const;
                        const FormatSettings: TFormatSettings) : Cardinal
```

Visibility: default

Description: `WideFormatBuf` calls simply `WideFormat` (1797) with `Fmt` (with length `FmtLen` bytes) and stores maximum `BufLen` bytes in the buffer `buf`. It returns the number of copied bytes.

See also: `WideFmtStr` (1797), `WideFormat` (1797), `Format` (1715), `FormatBuf` (1722)

76.15.324 WideLowerCase

Synopsis: Change a widestring to all-lowercase.

Declaration:

```
function WideLowerCase(const s: WideString) : WideString
```

Visibility: default

Description: `WideLowerCase` converts the string `S` to lowercase characters and returns the resulting string. It takes into account the operating system language settings when doing this, so special characters are converted correctly as well.

Remark On Unix-like platforms, a widestring manager must be installed for this function to work correctly.

Errors: None.

See also: `WideUpperCase` (1800)

76.15.325 WideSameStr

Synopsis: Check whether two widestrings are the same (case sensitive).

Declaration:

```
function WideSameStr(const s1: WideString; const s2: WideString)
                    : Boolean
```

Visibility: default

Description: `WideSameStr` returns `True` if `WideCompareStr` (1796) returns 0 (zero), i.e. when `S1` and `S2` are the same string (taking into account case).

See also: `WideSameText` (1798), `WideCompareStr` (1796), `WideCompareText` (1797), `AnsiSameStr` (1654)

76.15.326 WideSameText

Synopsis: Check whether two widestrings are the same (ignoring case).

Declaration:

```
function WideSameText(const s1: WideString; const s2: WideString)
                    : Boolean
```

Visibility: default

Description: `WideSameText` returns `True` if `WideCompareText` ([1797](#)) returns 0 (zero), i.e. when `S1` and `S2` are the same string (taking into account case).

See also: `WideSameStr` ([1798](#)), `WideCompareStr` ([1796](#)), `WideCompareText` ([1797](#)), `AnsiSameText` ([1654](#))

76.15.327 WideStrAlloc

Synopsis: Allocate a null-terminated widestring on the heap.

Declaration: `function WideStrAlloc(size: Cardinal) : PWideChar`

Visibility: default

Description: `WideStrAlloc` reserves memory on the heap for a widestring with length `Len`, (terminating `#0#0` included), and returns a pointer to it.

Additionally, `WideStrAlloc` allocates 4 extra bytes to store the size of the allocated memory.

Errors: None.

See also: `StrBufSize` ([1753](#)), `StrDispose` ([1756](#)), `StrAlloc` ([1283](#))

76.15.328 WideStringOf

Synopsis: Create Unicode string from array of bytes.

Declaration: `function WideStringOf(const Value: TBytes) : UnicodeString`

Visibility: default

Description: `WideStringOf` converts an array of bytes (`Bytes`) to a Unicode string. It considers each pair of bytes in the array as a single wide char. The array should have an even length. If the length is uneven, the last byte will be ignored.

This function performs the opposite operation of `BytesOf` ([1665](#)).

To create a string where the bytes are interpreted as ansichars, use `StringOf` ([1758](#)) instead.

See also: `StringOf` ([1758](#)), `WideBytesOf` ([1796](#)), `BytesOf` ([1665](#))

76.15.329 WideStringReplace

Synopsis: Replace occurrences of one substring with another in a widestring.

Declaration:

```
function WideStringReplace(const S: WideString;
                           const OldPattern: WideString;
                           const NewPattern: WideString;
                           Flags: TReplaceFlags) : WideString
function WideStringReplace(const S: WideString;
                           const OldPattern: WideString;
                           const NewPattern: WideString;
                           Flags: TReplaceFlags; out aCount: Integer)
                           : WideString
```

Visibility: default

Description: `WideStringReplace` searches the string `S` for occurrences of the string `OldPattern` and, if it is found, replaces it with `NewPattern`. It returns the resulting string. The behaviour of `StringReplace` can be tuned with `Flags`, which is of type `TReplaceFlags` ([1637](#)). Standard behaviour is to replace only the first occurrence of `OldPattern`, and to search case sensitively.

Errors: None.

See also: [TReplaceFlags \(1637\)](#), [StringReplace \(1759\)](#)

76.15.330 WideString

Synopsis: Change a widestring to all-uppercase.

Declaration: `function WideString(const s: WideString) : WideString`

Visibility: default

Description: `WideUpperCase` converts the string `S` to uppercase characters and returns the resulting string. It takes into account the operating system language settings when doing this, so special characters are converted correctly as well.

Remark On Unix-like platforms, a widestring manager must be installed for this function to work correctly.

Errors: None.

See also: [WideLowerCase \(1798\)](#)

76.15.331 WrapText

Synopsis: Word-wrap a text.

Declaration: `function WrapText(const Line: string; const BreakStr: string;
const BreakChars: TSysCharSet; MaxCol: Integer)
: string
function WrapText(const Line: string; MaxCol: Integer) : string`

Visibility: default

Description: `WrapText` does a wordwrap at column `MaxCol` of the string in `Line`. It breaks the string only at characters which are in `BreakChars` (default whitespace and hyphen) and inserts then the string `BreakStr` (default the lineending character for the current OS).

Portions of text that are between single or double quotes will be considered single words and the text will not be broken between the quotes.

See also: [StringReplace \(1759\)](#)

76.16 TDateTimeInfoRec

```
TDateTimeInfoRec = record
private
    data : tstatxplatform;
    function
        GetCreationTime : TDateTime;
        GetLastAccessTime : TDateTime
    ;
    function GetTimeStamp : TDateTime;
public
    property CreationTime
        : TDateTime;
    property LastAccessTime : TDateTime;
```

```

    property TimeStamp
    : TDateTime;
end

```

76.16.1 Property overview

Page	Properties	Access	Description
1801	CreationTime	r	
1801	LastAccessTime	r	
1801	TimeStamp	r	

76.16.2 TDateTimeInfoRec.CreationTime

Declaration: Property CreationTime : TDateTime

Visibility: public

Access: Read

76.16.3 TDateTimeInfoRec.LastAccessTime

Declaration: Property LastAccessTime : TDateTime

Visibility: public

Access: Read

76.16.4 TDateTimeInfoRec.TimeStamp

Declaration: Property TimeStamp : TDateTime

Visibility: public

Access: Read

76.17 TFormatSettings

```

TFormatSettings = record
public
    CurrencyFormat : Byte;
    NegCurrFormat
    : Byte;
    ThousandSeparator : AnsiChar;
    DecimalSeparator : AnsiChar
    ;
    CurrencyDecimals : Byte;
    DateSeparator : AnsiChar;
    TimeSeparator
    : AnsiChar;
    ListSeparator : AnsiChar;
    CurrencyString : string

```

```

;
ShortDateFormat : string;
LongDateFormat : string;
TimeAMString
: string;
TimePMString : string;
ShortTimeFormat : string;
LongTimeFormat : string;
ShortMonthNames : TMonthNameArray;
LongMonthNames
: TMonthNameArray;
ShortDayNames : TWeekNameArray;
LongDayNames
: TWeekNameArray;
TwoDigitYearCenturyWindow : Word;
class function
Invariant : TFormatSettings; Static;
Create;
end

```

TFormatSettings is a record that contains a copy of all variables which determine formatting in the various string formatting routines. It is used to pass local copies of these values to the various formatting routines in a thread-safe way.

76.17.1 Method overview

Page	Method	Description
1802	Create	
1802	Invariant	

76.17.2 TFormatSettings.Invariant

Declaration: class function Invariant : TFormatSettings; Static

Visibility: public

76.17.3 TFormatSettings.Create

Declaration: class function Create : TFormatSettings; Overload; Static
class function Create(const LocaleName: string) : TFormatSettings
; Overload; Static

Visibility: public

76.18 TOSVersion

```

TOSVersion = record
public
  TArchitecture = (arIntelX86, arIntelX64
, arARM32, arARM64, arIntelX16,
arXTensa, arAVR, arM68k

```

```

, arPowerPC, arPower64, arMips,
    arMipsel, arMips64
, arMips64el, arJVM, arWasm32, arSparc,
    arSparc64, arRiscV32
, arRiscV64, arZ80, arLoongArch64,
    arOther);
TPlatform
= (pfWindows, pfMacOS, pfiOS, pfAndroid, pfWinRT, pfLinux, pfGo32v2,
pfOS2, pfFreeBSD, pfBeos, pfNetBSD, pfAmiga, pfAtari, pfSolaris
,
    pfQNX, pfNetware, pfOpenBSD, pfWDosX, pfPalmOS, pfMacOSClassic
,
    pfDarwin, pfEMX, pfWatcom, pfMorphos, pfNetwLibC, pfWinCE
, pfGBA,
    pfNDS, pfEmbedded, pfSymbian, pfHaiku, pfIPhoneSim
, pfAIX,
    pfJava, pfNativeNT, pfMSDos, pfWII, pfAROS, pfDragonFly
, pfWin16,
    pfFreeRTOS, pfZX Spectrum, pfMSXDOS, pfAmstradCPC
, pfSinclairQL,
    pfWasi, pfOther);
TDelphiPlatform
= pfWindows..TPlatform.pfLinux;
AllArchitectures = [Low(TArchitecture)
)..High(TArchitecture)];
AllPlatforms = [Low(TPlatform)..High(TPlatform)
]);
private
    FFull : string;
    FArchitecture : TArchitecture;
    FBuild
    : Integer;
    FMajor : Integer;
    FMinor : Integer;
    FName : string
    ;
    FPlatform : TPlatform;
    FServicePackMajor : Integer;
    FServicePackMinor
    : Integer;
    FPrettyName : string;
    FLibCVersionMajor : Integer
    ;
    FLibCVersionMinor : Integer;
    class constructor Create;
public
    Check;
    class function ToString : string; Static;
    property
    Architecture : TArchitecture;
    property Build : Integer;
    property
    Major : Integer;
    property Minor : Integer;

```


76.18.5 TOSVersion.Architecture

Declaration: `Property Architecture : TArchitecture`

Visibility: public

Access: Read

76.18.6 TOSVersion.Build

Declaration: `Property Build : Integer`

Visibility: public

Access: Read

76.18.7 TOSVersion.Major

Declaration: `Property Major : Integer`

Visibility: public

Access: Read

76.18.8 TOSVersion.Minor

Declaration: `Property Minor : Integer`

Visibility: public

Access: Read

76.18.9 TOSVersion.Name

Declaration: `Property Name : string`

Visibility: public

Access: Read

76.18.10 TOSVersion.Platform

Declaration: `Property Platform : TPlatform`

Visibility: public

Access: Read

76.18.11 TOSVersion.ServicePackMajor

Declaration: `Property ServicePackMajor : Integer`

Visibility: public

Access: Read

76.18.12 TOSVersion.ServicePackMinor

Declaration: Property ServicePackMinor : Integer

Visibility: public

Access: Read

76.18.13 TOSVersion.PrettyName

Declaration: Property PrettyName : string

Visibility: public

Access: Read

76.18.14 TOSVersion.LibCVersionMajor

Declaration: Property LibCVersionMajor : Integer

Visibility: public

Access: Read

76.18.15 TOSVersion.LibCVersionMinor

Declaration: Property LibCVersionMinor : Integer

Visibility: public

Access: Read

76.19 TRawbyteSearchRec

```
TRawbyteSearchRec = record
public
    Time : Int64deprecated;
    Size
    : Int64;
    Attr : LongInt;
    Name : RawByteString;
    ExcludeAttr
    : LongInt;
    FindHandle : Pointer;
    Mode : TMode;
private
    function
    GetTimeStamp : TDateTime;
    function GetTimeStampUTC : TDateTime
    ;
public
    function IsDirectory : Boolean;
    function IsCurrentOrParentDir
    : Boolean;
```

```

property TimeStamp : TDateTime;
property TimeStampUTC
: TDateTime;
end

```

`TRawbyteSearchRec` is a search handle description record using single-byte strings. It is initialized by a call to `FindFirst` (1706) and can be used to do subsequent calls to `FindNext` (1708). It contains the result of these function calls. It must be used to close the search sequence with a call to `FindClose` (1706).

Remark Not all fields of this record should be used. Some of the fields are for internal use only. (`PathOnly` for example, is only provided for Kylix compatibility)

Remark Note that for files with Unicode filenames this is a converted value from the Unicode filename. Depending on the codepage, this may or may not be a correct rendering of the correct Unicode filename.

76.19.1 Method overview

Page	Method	Description
1807	<code>IsCurrentOrParentDir</code>	
1807	<code>IsDirectory</code>	

76.19.2 Property overview

Page	Properties	Access	Description
1807	<code>TimeStamp</code>	r	
1808	<code>TimeStampUTC</code>	r	

76.19.3 `TRawbyteSearchRec.IsDirectory`

Declaration: `function IsDirectory : Boolean`

Visibility: public

76.19.4 `TRawbyteSearchRec.IsCurrentOrParentDir`

Declaration: `function IsCurrentOrParentDir : Boolean`

Visibility: public

76.19.5 `TRawbyteSearchRec.TimeStamp`

Synopsis:

Declaration: `Property TimeStamp : TDateTime`

Visibility: public

Access: Read

Description: `TimeStamp` is the file timestamp (Date) in `TDateTime` format, as opposed to file system timestamp.

See also: `TRawbyteSearchRec.Date` (1807)

76.19.6 TRawbyteSearchRec.TimeStampUTC

Declaration: Property TimeStampUTC : TDateTime

Visibility: public

Access: Read

76.20 TRawbyteSymLinkRec

```
TRawbyteSymLinkRec = record
public
  TargetName : RawByteString;
  Size : Int64;
  Attr : LongInt;
  Mode : TMode;
private
  function
    GetTimeStamp : TDateTime;
public
  property TimeStamp : TDateTime
    ;
end
```

TSymLinkRec contains information about the target of a symlink. It is used in FileGetSymLink-Target (1700) to return information. The following fields are present:

TargetName Target file filename in single-byte string.

Size Target file size.

Attr Target file attributes.

Mode #ShortDescr:TRawbyteSymLinkRec.Mode

TimeStamp Target file timestamp.

76.20.1 Property overview

Page	Properties	Access	Description
1808	TimeStamp	r	Target file timestamp.

76.20.2 TRawbyteSymLinkRec.TimeStamp

Synopsis: Target file timestamp.

Declaration: Property TimeStamp : TDateTime

Visibility: public

Access: Read

76.21 TUnicodeSearchRec

```

TUnicodeSearchRec = record
public
    Time : Int64deprecated;
    Size
      : Int64;
    Attr : LongInt;
    Name : UnicodeString;
    ExcludeAttr
      : LongInt;
    FindHandle : Pointer;
    Mode : TMode;
private
    function
      GetTimeStamp : TDateTime;
    function GetTimeStampUTC : TDateTime
      ;
public
    function IsDirectory : Boolean;
    function IsCurrentOrParentDir
      : Boolean;
    property TimeStamp : TDateTime;
    property TimeStampUTC
      : TDateTime;
end

```

TRawbyteSearchRec is a search handle description record using multi-byte strings. It is initialized by a call to FindFirst (1706) and can be used to do subsequent calls to FindNext (1708). It contains the result of these function calls. It must be used to close the search sequence with a call to FindClose (1706).

Remark Not all fields of this record should be used. Some of the fields are for internal use only. (PathOnly for example, is only provided for Kylix compatibility)

76.21.1 Method overview

Page	Method	Description
1810	IsCurrentOrParentDir	
1809	IsDirectory	

76.21.2 Property overview

Page	Properties	Access	Description
1810	TimeStamp	r	Time stamp as datetime.
1810	TimeStampUTC	r	

76.21.3 TUnicodeSearchRec.IsDirectory

Declaration: function IsDirectory : Boolean

Visibility: public

76.21.4 TUnicodeSearchRec.IsCurrentOrParentDir

Declaration: `function IsCurrentOrParentDir : Boolean`

Visibility: `public`

76.21.5 TUnicodeSearchRec.TimeStamp

Synopsis: Time stamp as datetime.

Declaration: `Property TimeStamp : TDateTime`

Visibility: `public`

Access: `Read`

Description: `TimeStamp` is the file timestamp (Date) in `TDateTime` format, as opposed to file system timestamp.

See also: `TRawbyteSearchRec.Date` ([1807](#))

76.21.6 TUnicodeSearchRec.TimeStampUTC

Declaration: `Property TimeStampUTC : TDateTime`

Visibility: `public`

Access: `Read`

76.22 TUnicodeSymLinkRec

```
TUnicodeSymLinkRec = record
public
    TargetName : UnicodeString;
    Attr : LongInt;
    Size : Int64;
    Mode : TMode;
private
    function
        GetTimeStamp : TDateTime;
public
    property TimeStamp : TDateTime
        ;
end
```

`TUnicodeSymLinkRec` contains information about the target of a symlink. It is used in `FileGetSymLinkTarget` ([1700](#)) to return information. The following fields are present:

TargetName Target file filename in single-byte string.

Size Target file size.

Attr Target file attributes.

Mode `#ShortDescr:TRawbyteSymLinkRec.Mode`

TimeStamp Target file timestamp.

76.22.1 Property overview

Page	Properties	Access	Description
1811	TimeStamp	r	Target file timestamp.

76.22.2 TUnicodeSymLinkRec.TimeStamp

Synopsis: Target file timestamp.

Declaration: `Property TimeStamp : TDateTime`

Visibility: `public`

Access: `Read`

76.23 EAbort

76.23.1 Description

`EAbort` is raised by the `Abort` ([1647](#)) procedure. It is not displayed in GUI applications, and serves only to immediately abort the current procedure, and return control to the main program loop.

See also: `Abort` ([1647](#))

76.24 EAbstractError

76.24.1 Description

`EAbstractError` is raised when an abstract error occurs, i.e. when an unimplemented abstract method is called.

76.25 EAccessViolation

76.25.1 Description

`EAccessViolation` is raised when the OS reports an Access Violation, i.e. when invalid memory is accessed.

See also: `EObjectCheck` ([1818](#))

76.26 EArgumentException

76.26.1 Description

`EArgumentException` is raised by many character conversion/handling routines to indicate an erroneous argument was passed to the function (usually indicating an invalid codepoint in a Unicode string).

See also: `EArgumentOutOfRangeException` ([1812](#))

76.27 EArgumentNilException

76.27.1 Description

`EArgumentNilException` is an exception raised when an argument is `Nil` when it should not be `Nil`.

This exception class is provided for Delphi compatibility, but is not actually used in FPC.

See also: `Exception` ([1821](#))

76.28 EArgumentOutOfRangeException

76.28.1 Description

`EArgumentOutOfRangeException` is raised by many character conversion/handling routines to indicate an erroneous argument was passed to the function (indicating an invalid character index in a Unicode string).

See also: `EArgumentException` ([1811](#))

76.29 EAssertionFailed

76.29.1 Description

`EAssertionFailed` is raised when an application that is compiled with assertions, encounters an invalid assertion.

76.30 EBusError

76.30.1 Description

`EBusError` is raised in case of a bus error.

76.31 ECFError

76.32 EControlC

76.32.1 Description

`EControlC` is raised when the user has pressed CTRL-C in a console application.

76.33 EConvertError

76.33.1 Description

`EConvertError` is raised by the various conversion routines in the `SysUtils` unit. The message will contain more specific error information.

76.34 EDirectoryNotFoundException

76.34.1 Description

`EDirectoryNotFoundException` is an exception raised when a directory is referenced that does not exist.

This exception class is provided for Delphi compatibility, but is not actually used in FPC.

See also: `Exception` ([1821](#)), `EFileNotFoundException` ([1813](#)), `EPathNotFoundException` ([1819](#)), `EPathTooLongException` ([1819](#))

76.35 EDivByZero

76.35.1 Description

`EDivByZero` is used when the operating system or CPU signals a division by zero error.

76.36 EEncodingError

76.36.1 Description

`EEncodingError` is the exception classed used by the `TEncoding` class to indicate errors.

See also: `TEncoding` ([1871](#))

76.37 EExternal

76.37.1 Description

`EExternal` is the base exception for all external exceptions, as reported by the CPU or operating system, as opposed to internal exceptions, which are raised by the program itself. The `SysUtils` unit converts all operating system errors to descendants of `EExternal`.

See also: `EInterror` ([1815](#)), `EExternal` ([1813](#)), `EMathError` ([1817](#)), `EExternalException` ([1813](#)), `EAccessViolation` ([1811](#)), `EPrivilege` ([1819](#)), `EStackOverflow` ([1820](#)), `EControlC` ([1812](#))

76.38 EExternalException

76.38.1 Description

`EExternalException` is raised when an external routine raises an exception.

See also: `EExternal` ([1813](#))

76.39 EFileNotFoundException

76.39.1 Description

`EFileNotFoundException` is an exception raised when a file is referenced that does not exist.

This exception class is provided for Delphi compatibility, but is not actually used in FPC.

See also: [Exception \(1821\)](#), [EPathNotFoundException \(1819\)](#), [EDirectoryNotFoundException \(1813\)](#), [EPathTooLongException \(1819\)](#)

76.40 EFormatError

76.40.1 Description

`EFormatError` is raised in case of an error in one of the various [Format \(1715\)](#) functions.

See also: [Format \(1715\)](#)

76.41 EHeapMemoryError

76.41.1 Description

`EHeapMemoryError` is raised when an error occurs in heap (dynamically allocated) memory.

See also: [EHeapException \(1625\)](#), [EoutOfMemory \(1818\)](#), [EInvalidPointer \(1816\)](#)

76.41.2 Method overview

Page	Method	Description
1814	<code>FreeInstance</code>	Free the exception instance.

76.41.3 EHeapMemoryError.FreeInstance

Synopsis: Free the exception instance.

Declaration: `procedure FreeInstance; Override`

Visibility: `public`

Description: `FreeInstance` checks whether the exception instance may be freed prior to calling the inherited `FreeInstance`. The exception is only freed in case of normal program shutdown, if a heap error occurred, the exception instance is not freed.

76.42 EInOutArgumentException

76.42.1 Method overview

Page	Method	Description
1814	<code>Create</code>	
1815	<code>CreateFmt</code>	
1815	<code>CreateRes</code>	

76.42.2 EInOutArgumentException.Create

Declaration: `constructor Create(const aMsg: string; const aPath: string); Overload`

Visibility: `public`

76.42.3 EInOutArgumentException.CreateRes

Declaration: constructor CreateRes(ResStringRec: PResStringRec; const aPath: string)
; Overload

Visibility: public

76.42.4 EInOutArgumentException.CreateFmt

Declaration: constructor CreateFmt(const fmt: string; const args: Array of const;
const aPath: string); Overload

Visibility: public

76.43 EInOutError**76.43.1 Description**

EInOutError is raised when a IO routine of Free Pascal returns an error. The error is converted to an EInOutError only if the input/output checking feature of FPC is turned on. The error code of the input/output operation is returned in ErrorCode (??).

See also: EInOutError.ErrorCode (??)

76.44 EIntError**76.44.1 Description**

EIntError is used when the operating system or CPU signals an integer operation error, e.g., an overflow.

76.45 EIntfCastError**76.45.1 Description**

EIntfCastError is raised when an invalid interface cast is encountered.

See also: EInvalidCast ([1816](#))

76.46 EIntOverflow**76.46.1 Description**

EIntOverflow is used when the operating system or CPU signals a integer overflow error.

See also: EIntError ([1815](#)), EDivByZero ([1813](#)), ERangeError ([1820](#))

76.47 EInvalidCast

76.47.1 Description

EInvalidCast is raised when an invalid typecast error (using the `as` operator) is encountered.

See also: EIntfCastError ([1815](#))

76.48 EInvalidContainer

76.48.1 Description

EInvalidContainer is not yet used by Free Pascal, and is provided for Delphi compatibility only.

76.49 EInvalidInsert

76.49.1 Description

EInvalidInsert is not yet used by Free Pascal, and is provided for Delphi compatibility only.

76.50 EInvalidOp

76.50.1 Description

EInvalidOp is raised when an invalid operation is encountered.

76.51 EInvalidOpException

76.51.1 Description

EInvalidOpException is raised when an invalid operation is attempted.

See also: Exception ([1821](#))

76.52 EInvalidPointer

76.52.1 Description

EInvalidPointer is raised when an invalid heap pointer is used.

See also: EHeapException ([1625](#)), EHeapMemoryError ([1814](#)), EOutOfMemory ([1818](#))

76.53 EListError

76.54 EMathError

76.54.1 Description

`EMathError` is used when the operating system or CPU signals a floating point overflow error.

See also: `EIntError` ([1815](#)), `EIntOverflow` ([1815](#)), `EDivByZero` ([1813](#)), `ERangeError` ([1820](#))

76.55 EMonitor

76.56 EMonitorLockException

76.57 ENoConstructException

76.57.1 Description

`ENoConstructException` is the exception raised when an instance of type `TCharacter` is being created. The `TCharacter` class only contains static methods, no instances of this class should be instantiated.

76.58 ENoDynLibsSupport

76.58.1 Description

`ENoDynLibsSupport` is raised when no dynamic library support is detected and a dynamic library loading operation is attempted.

See also: `EThreadError` ([1821](#)), `Exception` ([1821](#))

76.59 ENoMonitorSupportException

76.60 ENoThreadSupport

76.60.1 Description

`ENoThreadSupport` is raised when some thread routines are invoked, and thread support was not enabled when the program was compiled.

76.61 ENotImplemented

76.61.1 Description

`ENotImplemented` can be used to raise an exception when a particular call had been defined, but was not implemented.

76.62 ENotSupportedException

76.62.1 Description

`ENotSupportedException` is an exception raised when a function or procedure is not supported for a certain platform.

This exception class is provided for Delphi compatibility, but is not actually used in FPC.

See also: `Exception` ([1821](#))

76.63 ENoWideStringSupport

76.63.1 Description

`ENoWideStringSupport` is the exception raised when a run-time 233 occurs, i.e. when widestring routines are called and the application does not contain widestring support.

76.64 EObjectCheck

76.64.1 Description

`EObjectCheck` is raised when the `-CR` (check object references) command-line option or `{ $OBJECTCHECKSON }` directive is in effect and a `Nil` reference to an object or class was encountered.

See also: `EAccessViolation` ([1811](#))

76.65 EObjectDisposed

76.66 EOperationCancelled

76.67 EOSError

76.67.1 Description

`EOSError` is raised when some Operating System call fails. The `ErrorCode` (??) property contains the operating system error code.

See also: `EOSError.ErrorCode` (??)

76.68 EOutOfMemory

76.68.1 Description

`EOutOfMemory` occurs when memory can no longer be allocated on the heap. An instance of `EOutOfMemory` is allocated on the heap at program startup, so it is available when needed.

See also: `EHeapException` ([1625](#)), `EHeapMemoryError` ([1814](#)), `EInvalidPointer` ([1816](#))

76.69 EOverflow

76.69.1 Description

`EOverflow` occurs when a float operation overflows. (i.e. result is too big to represent).

See also: `EIntError` ([1815](#)), `EIntOverflow` ([1815](#)), `EDivByZero` ([1813](#)), `ERangeError` ([1820](#)), `EUnderFlow` ([1821](#))

76.70 EPackageError

76.70.1 Description

`EPackageError` is not yet used by Free Pascal, and is provided for Delphi compatibility only.

76.71 EPathNotFoundException

76.71.1 Description

`EPathNotFoundException` is an exception raised when a path is referenced that does not exist. This exception class is provided for Delphi compatibility, but is not actually used in FPC.

See also: `Exception` ([1821](#)), `EFileNotFoundException` ([1813](#)), `EDirectoryNotFoundException` ([1813](#)), `EPathTooLongException` ([1819](#))

76.72 EPathTooLongException

76.72.1 Description

`EPathTooLongException` is an exception raised when a pathname argument is longer than the allowed pathname length for files.

This exception class is provided for Delphi compatibility, but is not actually used in FPC.

See also: `Exception` ([1821](#)), `EFileNotFoundException` ([1813](#)), `EDirectoryNotFoundException` ([1813](#)), `EPathNotFoundException` ([1819](#))

76.73 EPrivilege

76.73.1 Description

`EPrivilege` is raised when the OS reports that an invalid instruction was executed.

76.74 EProgrammerNotFound

76.74.1 Description

`EProgrammerNotFound` is a Delphi-compatibility exception. It is not used in FPC. Some replies on [stackoverflow](#) suggest it is an internal joke of the Delphi team.

76.75 EPropReadOnly

76.75.1 Description

`EPropReadOnly` is raised when an attempt is made to write to a read-only property.

76.76 EPropWriteOnly

76.76.1 Description

`EPropWriteOnly` is raised when an attempt is made to read from a write-only property.

See also: `EPropReadOnly` ([1820](#))

76.77 ERangeError

76.77.1 Description

`ERangeError` is raised by the Free Pascal runtime library if range checking is on, and a range check error occurs.

See also: `EIntError` ([1815](#)), `EDivByZero` ([1813](#)), `EIntOverflow` ([1815](#))

76.78 ESafecallException

76.78.1 Description

`ESafecallException` is not yet used by Free Pascal, and is provided for Delphi compatibility only.

76.79 ESigQuit

76.79.1 Description

`ESigQuit` is used when a QUIT signal is received and a run-time error 233 is converted to an exception.

See also: `Exception` ([1821](#))

76.80 EStackOverflow

76.80.1 Description

`EStackOverflow` occurs when the stack has grown too big (e.g. by infinite recursion).

76.81 EThreadError

76.82 EUnderflow

76.82.1 Description

EOverflow occurs when a float operation underflows (i.e. result is too small to represent).

See also: EIntError ([1815](#)), EIntOverflow ([1815](#)), EDivByZero ([1813](#)), ERangeError ([1820](#)), EOverflow ([1819](#))

76.83 EVariantError

76.83.1 Description

EVariantError is raised by the internal variant routines.

76.83.2 Method overview

Page	Method	Description
1821	CreateCode	Create an instance of EVariantError with a particular error code.

76.83.3 EVariantError.CreateCode

Synopsis: Create an instance of EVariantError with a particular error code.

Declaration: constructor CreateCode (Code: LongInt)

Visibility: default

Description: CreateCode calls the inherited constructor, and sets the ErrCode (??) property to Code.

See also: ErrCode (??)

76.84 Exception

76.84.1 Description

Exception is the base class for all exception handling routines in the RTL and FCL. While it is possible to raise an exception with any class descending from TObject, it is recommended to use Exception as the basis of exception class objects: the Exception class introduces properties to associate a message and a help context with the exception being raised. What is more, the SysUtils unit sets the necessary hooks to catch and display unhandled exceptions: in such cases, the message displayed to the end user, will be the message stored in the exception class.

See also: ExceptObject ([1685](#)), ExceptAddr ([1684](#)), ExceptionErrorMessage ([1685](#)), ShowException ([1751](#)), Abort ([1647](#))

76.84.2 Method overview

Page	Method	Description
1822	Create	Constructs a new exception object with a given message.
1822	CreateFmt	Constructs a new exception object and formats a new message.
1823	CreateFmtHelp	Constructs a new exception object and sets the help context and formats the message.
1823	CreateHelp	Constructs a new exception object and sets the help context.
1822	CreateRes	Constructs a new exception object and gets the message from a resource.
1823	CreateResFmt	Constructs a new exception object and formats the message from a resource.
1824	CreateResFmtHelp	Constructs a new exception object and sets the help context and formats the message from a resource.
1823	CreateResHelp	Constructs a new exception object and sets the help context and gets the message from a resource.
1824	GetBaseException	
1824	ToString	Nicely formatted version of the exception message.

76.84.3 Property overview

Page	Properties	Access	Description
1824	HelpContext	rw	Help context associated with the exception.
1824	Message	rw	Message associated with the exception.

76.84.4 Exception.Create

Synopsis: Constructs a new exception object with a given message.

Declaration: `constructor Create(const msg: string)`

Visibility: public

Errors: Construction may fail if there is not enough memory on the heap.

See also: `Exception.CreateFmt` ([1822](#)), `Exception.Message` ([1824](#))

76.84.5 Exception.CreateFmt

Synopsis: Constructs a new exception object and formats a new message.

Declaration: `constructor CreateFmt(const msg: string; const args: Array of const)`

Visibility: public

Errors: Construction may fail if there is not enough memory on the heap.

See also: `Exception.Create` ([1822](#)), `Exception.Message` ([1824](#)), `Format` ([1715](#))

76.84.6 Exception.CreateRes

Synopsis: Constructs a new exception object and gets the message from a resource.

Declaration: `constructor CreateRes(ResString: PString)`

Visibility: public

Errors: Construction may fail if there is not enough memory on the heap.

See also: `Exception.Create` (1822), `Exception.CreateFmt` (1822), `Exception.CreateResFmt` (1823), `Exception.Message` (1824)

76.84.7 `Exception.CreateResFmt`

Synopsis: Constructs a new exception object and formats the message from a resource.

Declaration: `constructor CreateResFmt(ResString: PString; const Args: Array of const)`

Visibility: public

Description: `CreateResFmt` does the same as `CreateFmt` (1822), but fetches the message from the resource string `ResString`.

Errors: Construction may fail if there is not enough memory on the heap.

See also: `Exception.Create` (1822), `Exception.CreateFmt` (1822), `Exception.CreateRes` (1822), `Exception.Message` (1824)

76.84.8 `Exception.CreateHelp`

Synopsis: Constructs a new exception object and sets the help context.

Declaration: `constructor CreateHelp(const Msg: string; AHelpContext: LongInt)`

Visibility: public

Description: `CreateHelp` does the same as the `Create` (1822) constructor, but additionally stores `AHelpContext` in the `HelpContext` (1824) property.

See also: `Exception.Create` (1822)

76.84.9 `Exception.CreateFmtHelp`

Synopsis: Constructs a new exception object and sets the help context and formats the message.

Declaration: `constructor CreateFmtHelp(const Msg: string;
const Args: Array of const;
AHelpContext: LongInt)`

Visibility: public

Description: `CreateFmtHelp` does the same as the `CreateFmt` (1822) constructor, but additionally stores `AHelpContext` in the `HelpContext` (1824) property.

See also: `Exception.CreateFmt` (1822)

76.84.10 `Exception.CreateResHelp`

Synopsis: Constructs a new exception object and sets the help context and gets the message from a resource.

Declaration: `constructor CreateResHelp(ResString: PString; AHelpContext: LongInt)`

Visibility: public

Description: `CreateResHelp` does the same as the `CreateRes` (1822) constructor, but additionally stores `AHelpContext` in the `HelpContext` (1824) property.

See also: `Exception.CreateRes` (1822)

76.84.11 Exception.CreateResFmtHelp

Synopsis: Constructs a new exception object and sets the help context and formats the message from a resource.

Declaration: `constructor CreateResFmtHelp (ResString: PString;
const Args: Array of const;
AHelpContext: LongInt)`

Visibility: public

Description: `CreateResFmtHelp` does the same as the `CreateResFmt` (1823) constructor, but additionally stores `AHelpContext` in the `HelpContext` (1824) property.

See also: `Exception.CreateResFmt` (1823)

76.84.12 Exception.ToString

Synopsis: Nicely formatted version of the exception message.

Declaration: `function ToString : string; Override`

Visibility: public

Description: `ToString` overrides the `ToString` method to return a concatenation of classname and `Exception.Message` (1824).

See also: `Exception.Message` (1824)

76.84.13 Exception.GetBaseException

Declaration: `function GetBaseException : Exception; Virtual`

Visibility: public

76.84.14 Exception.HelpContext

Synopsis: Help context associated with the exception.

Declaration: `Property HelpContext : LongInt`

Visibility: public

Access: Read,Write

Description: `HelpContext` is the help context associated with the exception, and can be used to provide context-sensitive help when the exception error message is displayed. It should be set in the exception constructor.

See also: `Exception.CreateHelp` (1823), `Exception.Message` (1824)

76.84.15 Exception.Message

Synopsis: Message associated with the exception.

Declaration: `Property Message : string`

Visibility: public

Access: Read,Write

Description: `Message` provides additional information about the exception. It is shown to the user in e.g. the `ShowException` (1751) routine, and should be set in the constructor when the exception is raised.

See also: `Exception.Create` (1822), `Exception.HelpContext` (1824)

76.85 EZeroDivide

76.85.1 Description

`EZeroDivide` occurs when a float division by zero occurs.

See also: `EIntError` (1815), `EIntOverflow` (1815), `EDivByZero` (1813), `ERangeError` (1820)

76.86 IReadWriteSync

76.86.1 Description

`IReadWriteSync` is an interface for synchronizing read/write operations. Writers are always guaranteed to have exclusive access: readers may or may not have simultaneous access, depending on the implementation.

76.86.2 Method overview

Page	Method	Description
1825	<code>BeginRead</code>	Start a read operation.
1826	<code>BeginWrite</code>	Start a write operation.
1825	<code>EndRead</code>	End a read operation.
1826	<code>EndWrite</code>	End a write operation.

76.86.3 IReadWriteSync.BeginRead

Synopsis: Start a read operation.

Declaration: `procedure BeginRead`

Visibility: default

Description: `BeginRead` indicates that a read operation is about to be started. If a write operation is in progress, then the call will block until the write operation finished. Depending on the implementation the call may also block if another read operation is in progress.

After `BeginRead`, any write operation started with `BeginWrite` (1826) will block until `EndRead` (1825) is called.

See also: `IReadWriteSync.EndRead` (1825), `IReadWriteSync.BeginWrite` (1826), `IReadWriteSync.EndWrite` (1826)

76.86.4 IReadWriteSync.EndRead

Synopsis: End a read operation.

Declaration: `procedure EndRead`

Visibility: default

Description: `EndRead` signals the end of a read operation. If there was any blocked write operation, that will be unblocked by a call to `EndRead`.

See also: `IReadWriteSync.BeginRead` (1825), `IReadWriteSync.BeginWrite` (1826), `IReadWriteSync.EndWrite` (1826)

76.86.5 `IReadWriteSync.BeginWrite`

Synopsis: Start a write operation.

Declaration: `function BeginWrite : Boolean`

Visibility: default

Description: `BeginWrite` signals the begin of a write operation. This call will block if any other read or write operation is currently in progress. It will resume only after all other read or write operations have finished.

See also: `IReadWriteSync.EndRead` (1825), `IReadWriteSync.EndWrite` (1826), `IReadWriteSync.BeginRead` (1825)

76.86.6 `IReadWriteSync.EndWrite`

Synopsis: End a write operation.

Declaration: `procedure EndWrite`

Visibility: default

Description: `EndWrite` signals the end of a write operation. After the call to `EndWrite` any other read or write operations can start.

See also: `IReadWriteSync.EndRead` (1825), `IReadWriteSync.EndWrite` (1826), `IReadWriteSync.BeginRead` (1825)

76.87 TANSISTRINGBUILDER

76.87.1 Description

`TAnsiStringBuilder` is a class to construct an ansistring out of other strings or characters. It's methods can be used instead of constructing the string manually using the normal + or concat operators. If used properly (e.g. by setting a sufficiently large `Capacity` (1833)), will result in faster operation. It offers various methods to append to the string.

To create a unicode string, use the `TUnicodeStringBuilder` (1963) class instead.

See also: `TUnicodeStringBuilder` (1963)

76.87.2 Method overview

Page	Method	Description
1827	Append	Append data in string form to the buffer.
1828	AppendFormat	Append the result of a <code>Format</code> .
1829	AppendLine	Append a string followed by a newline.
1829	Clear	Clear the string being built.
1829	CopyTo	Copy string bytes to an array.
1827	Create	Create a new <code>TAnsiStringBuilder</code> instance.
1829	EnsureCapacity	Set the capacity if needed.
1830	Equals	Check if 2 stringbuilders have the same content string.
1830	GetEnumerator	
1830	Insert	Insert data in string form into the buffer at a given position.
1831	Remove	Remove data from the string.
1831	Replace	Replace a range of characters.
1832	ToString	Return the string buffer as an <code>AnsiString</code> .

76.87.3 Property overview

Page	Properties	Access	Description
1833	Capacity	rw	Current capacity of the string buffer.
1832	Chars	rw	Indexed access to the characters in the string.
1832	Length	rw	Current length of the string buffer.
1833	MaxCapacity	r	Maximum capacity of the string buffer.

76.87.4 TANSISTRINGBUILDER.Create

Synopsis: Create a new `TAnsiStringBuilder` instance.

Declaration: constructor `Create`

```

constructor Create(aCapacity: Integer)
constructor Create(const AValue: ANSISTRING)
constructor Create(aCapacity: Integer; aMaxCapacity: Integer)
constructor Create(const AValue: ANSISTRING; aCapacity: Integer)
constructor Create(const AValue: ANSISTRING; StartIndex: Integer;
                  aLength: Integer; aCapacity: Integer)

```

Visibility: public

Description: `Create` creates an empty `TAnsiStringBuilder` instance. It can optionally set the initial capacity to `aCapacity` and the maximum capacity to `aMaxCapacity` and can also initialize the string buffer with characters from `aValue`; `aLength` characters starting at `StartIndex` (1-based) will be copied to the string buffer. If `aLength` and `StartIndex` are omitted, then all characters are copied.

See also: [Capacity \(1833\)](#), [MaxCapacity \(1833\)](#)

76.87.5 TANSISTRINGBUILDER.Append

Synopsis: Append data in string form to the buffer.

Declaration: function `Append(const AValue: Boolean) : TANSISTRINGBUILDER`
function `Append(const AValue: Byte) : TANSISTRINGBUILDER`
function `Append(const AValue: AnsiChar) : TANSISTRINGBUILDER`
function `Append(const AValue: Currency) : TANSISTRINGBUILDER`

```

function Append(const AValue: Double) : TANSISTRINGBUILDER
function Append(const AValue: SmallInt) : TANSISTRINGBUILDER
function Append(const AValue: LongInt) : TANSISTRINGBUILDER
function Append(const AValue: Int64) : TANSISTRINGBUILDER
function Append(const AValue: TObject) : TANSISTRINGBUILDER
function Append(const AValue: ShortInt) : TANSISTRINGBUILDER
function Append(const AValue: Single) : TANSISTRINGBUILDER
function Append(const AValue: UInt64) : TANSISTRINGBUILDER
function Append(const AValue: Array of AnsiChar) : TANSISTRINGBUILDER
function Append(const AValue: Word) : TANSISTRINGBUILDER
function Append(const AValue: Cardinal) : TANSISTRINGBUILDER
function Append(const AValue: PAnsiChar) : TANSISTRINGBUILDER
function Append(const AValue: RawByteString) : TANSISTRINGBUILDER
function Append(const AValue: AnsiChar; RepeatCount: Integer)
    : TANSISTRINGBUILDER
function Append(const AValue: Array of AnsiChar; StartIndex: Integer;
    SBCharCount: Integer) : TANSISTRINGBUILDER
function Append(const AValue: ANSISTRING; StartIndex: Integer;
    Count: Integer) : TANSISTRINGBUILDER
function Append(const Fmt: ANSISTRING; const Args: Array of const)
    : TANSISTRINGBUILDER

```

Visibility: public

Description: Append in its various overloaded forms will append the `aValue` argument to the buffer, after converting it to a string value if necessary:

- For integer values the various `IntToStr` (1738) functions will be used.
- For `TObject` (1609) the `TObject.ToString` (1609) is used.
- The `repeatCount` argument will repeat the character as many times as instructed using `StringOfChar` (1609).
- If the `StartIndex` and `SBCharCount` are specified, then `SBCharCount` characters are copied starting from position `StartIndex`.

An overloaded version exists with the same arguments as `AppendFormat` (1828).

See also: `StringOfChar` (1609), `AppendFormat` (1828), `AppendLine` (1829), `Insert` (1830), `Remove` (1831)

76.87.6 TANSISTRINGBUILDER.AppendFormat

Synopsis: Append the result of a `Format`.

Declaration: `function AppendFormat(const Fmt: ANSISTRING; const Args: Array of const) : TANSISTRINGBUILDER`

Visibility: public

Description: `AppendFormat` calls `Format` (1715) with `fmt` and `Args` and calls `append` (1827) with the resulting string.

Errors: the call to `Format` may result in an exception if the format parameters do not match the passed arguments.

See also: `Append` (1827), `Insert` (1830), `Remove` (1831), `Format` (1715)

76.87.7 TANSISTRINGBUILDER.AppendLine

Synopsis: Append a string followed by a newline.

Declaration: `function AppendLine : TANSISTRINGBUILDER
function AppendLine(const AValue: RawByteString) : TANSISTRINGBUILDER`

Visibility: public

Description: `AppendLine` will append `aValue` (if it is specified) and appends the OS linebreak character (`sLineBreak` (1361)).

See also: `TAnsiStringBuilder.AppendFormat` (1828), `TAnsiStringBuilder.Append` (1827), `#rtl.system.sLineBreak` (1361), `Remove` (1831), `Insert` (1830)

76.87.8 TANSISTRINGBUILDER.Clear

Synopsis: Clear the string being built.

Declaration: `procedure Clear`

Visibility: public

Description: `Clear` sets the length of the string to 0 and sets the `Capacity` (1833) to the default capacity (64 bytes).

See also: `Capacity` (1833), `Remove` (1831)

76.87.9 TANSISTRINGBUILDER.CopyTo

Synopsis: Copy string bytes to an array.

Declaration: `procedure CopyTo(SourceIndex: Integer;
var Destination: Array of AnsiChar;
DestinationIndex: Integer; Count: Integer)`

Visibility: public

Description: `CopyTo` copies `aCount` characters from the string, starting at position `SourceIndex` (0-based) in the string to build, to the character array `Destination` at position `DestinationIndex`.

Errors: If the range of characters to copy falls outside the allowed range of characters in either source (available characters) or destination (available room for characters), then an `ERangeError` (1820) exception is raised.

See also: `ERangeError` (1820)

76.87.10 TANSISTRINGBUILDER.EnsureCapacity

Synopsis: Set the capacity if needed.

Declaration: `function EnsureCapacity(aCapacity: Integer) : Integer`

Visibility: public

Description: `EnsureCapacity` checks that `aCapacity` is in the allowed range (less than `MaxCapacity` (1833)) and if the current capacity is less than `aCapacity`, increases the capacity to `aCapacity`.

Errors: If `aCapacity` is larger than `MaxCapacity` (1833) or negative, an `ERangeError` (1820) exception is raised.

See also: `MaxCapacity` (1833)

76.87.11 TANSISTRINGBUILDER.Equals

Synopsis: Check if 2 stringbuilders have the same content string.

```
Declaration: function Equals(StringBuilder: TANSISTRINGBUILDER) : Boolean
                ; Reintroduce
```

Visibility: public

Description: `Equals` checks that the self instance has the same content as the `StringBuilder` instance. It checks the length and content (as bytes) of the strings.

See also: [TAnsiStringBuilder.Clear \(1829\)](#)

76.87.12 TANSISTRINGBUILDER.GetEnumerator

Declaration: function GetEnumerator : TCharEnumerator

Visibility: public

76.87.13 TANSISTRINGBUILDER.Insert

Synopsis: Insert data in string form into the buffer at a given position.

```

Declaration: function Insert(Index: Integer; const AValue: Boolean)
                : TANSISTRINGBUILDER
function Insert(Index: Integer; const AValue: Byte) : TANSISTRINGBUILDER
function Insert(Index: Integer; const AValue: AnsiChar)
                : TANSISTRINGBUILDER
function Insert(Index: Integer; const AValue: Currency)
                : TANSISTRINGBUILDER
function Insert(Index: Integer; const AValue: Double)
                : TANSISTRINGBUILDER
function Insert(Index: Integer; const AValue: SmallInt)
                : TANSISTRINGBUILDER
function Insert(Index: Integer; const AValue: LongInt)
                : TANSISTRINGBUILDER
function Insert(Index: Integer; const AValue: Array of AnsiChar)
                : TANSISTRINGBUILDER
function Insert(Index: Integer; const AValue: Int64)
                : TANSISTRINGBUILDER
function Insert(Index: Integer; const AValue: TObject)
                : TANSISTRINGBUILDER
function Insert(Index: Integer; const AValue: ShortInt)
                : TANSISTRINGBUILDER
function Insert(Index: Integer; const AValue: Single)
                : TANSISTRINGBUILDER
function Insert(Index: Integer; const AValue: ANSISTRING)
                : TANSISTRINGBUILDER
function Insert(Index: Integer; const AValue: Word) : TANSISTRINGBUILDER
function Insert(Index: Integer; const AValue: Cardinal)
                : TANSISTRINGBUILDER
function Insert(Index: Integer; const AValue: UInt64)
                : TANSISTRINGBUILDER
function Insert(Index: Integer; const AValue: ANSISTRING;

```

```

        const aRepeatCount: Integer) : TANSISTRINGBUILDER
function Insert(Index: Integer; const AValue: Array of AnsiChar;
               startIndex: Integer; SBCharCount: Integer)
               : TANSISTRINGBUILDER

```

Visibility: public

Description: Insert in its various overloaded forms will insert the aValue argument into the buffer at position Index (0-based), after converting it to a string value if necessary:

- For integer values the various IntToStr (1738) functions will be used.
- For TObject (1609) the TObject.ToString (1609) is used.
- The repeatCount argument will insert the character as many times as instructed.
- If the StartIndex and SBCharCount are specified, then SBCharCount characters are copied starting from position StartIndex.

See also: AppendFormat (1828), AppendLine (1829), Append (1827), Remove (1831), Replace (1831)

76.87.14 TANSISTRINGBUILDER.Remove

Synopsis: Remove data from the string.

```

Declaration: function Remove(StartIndex: Integer; RemLength: Integer)
               : TANSISTRINGBUILDER

```

Visibility: public

Description: Remove will remove RemLength characters from the string buffer, starting at position StartIndex (0-based).

Errors: If the range of characters to remove falls outside the allowed range of characters, then an ERangeError (1820) exception is raised.

See also: AppendFormat (1828), AppendLine (1829), Append (1827), Insert (1830), Clear (1829)

76.87.15 TANSISTRINGBUILDER.Replace

Synopsis: Replace a range of characters.

```

Declaration: function Replace(const OldChar: AnsiChar; const NewChar: AnsiChar)
               : TANSISTRINGBUILDER
function Replace(const OldChar: AnsiChar; const NewChar: AnsiChar;
               StartIndex: Integer; Count: Integer)
               : TANSISTRINGBUILDER
function Replace(const OldValue: RawByteString;
               const NewValue: RawByteString) : TANSISTRINGBUILDER
function Replace(const OldValue: RawByteString;
               const NewValue: RawByteString; StartIndex: Integer;
               Count: Integer) : TANSISTRINGBUILDER

```

Visibility: public

Description: Replace replaces occurrences of OldValue with the characters in NewValue. The search operation starts at position StartIndex (0-based, default 0) till position StartIndex+aCount is reached.

Errors: If `StartIndex` or `StartIndex+aCount` is out of range, an `ERangeError` (1820) exception is raised.

See also: `AppendFormat` (1828), `AppendLine` (1829), `Append` (1827), `Insert` (1830), `Clear` (1829), `Remove` (1831)

76.87.16 TANSISTRINGBUILDER.ToString

Synopsis: Return the string buffer as an `AnsiString`.

Declaration: `function ToString : ANSISTRING; Override`
`function ToString(aStartIndex: Integer; aLength: Integer) : ANSISTRING`
`; Reintroduce`

Visibility: public

Description: `ToString` returns the current content of the buffer as an `AnsiString`.

See also: `Clear` (1829), `CopyTo` (1829)

76.87.17 TANSISTRINGBUILDER.Chars

Synopsis: Indexed access to the characters in the string.

Declaration: `Property Chars[index: Integer]: AnsiChar; default`

Visibility: public

Access: Read,Write

Description: `Chars` allows indexed access to the characters in the string using a zero-based `Index`. The characters can be read or written. The allowed range for `Index` is 0 to `Length` (1832)-1.

See also: `Length` (1832)

76.87.18 TANSISTRINGBUILDER.Length

Synopsis: Current length of the string buffer.

Declaration: `Property &Length : Integer`

Visibility: public

Access: Read,Write

Description: `Length` can be used to get or set the current length (in characters) of the string buffer. When setting the length, the new length is checked whether it lies within the maximum capacity (`MaxCapacity` (1833)) for the string buffer: if not, an `ERangeError` (1820) exception is raised. The capacity of the buffer will be adjusted so the buffer has enough capacity to accomodate the new length.

See also: `MaxCapacity` (1833), `Capacity` (1833), `ERangeError` (1820)

76.87.19 TANSISTRINGBUILDER.Capacity

Synopsis: Current capacity of the string buffer.

Declaration: `Property Capacity : Integer`

Visibility: `public`

Access: `Read, Write`

Description: `Capacity` can be used to get or set the capacity of the string buffer (in characters). When setting the capacity, the new length is checked whether it lies within the maximum capacity (`MaxCapacity` (1833)) for the string buffer: if not, an `ERangeError` (1820) exception is raised.

See also: `MaxCapacity` (1833), `Length` (1832), `ERangeError` (1820)

76.87.20 TANSISTRINGBUILDER.MaxCapacity

Synopsis: Maximum capacity of the string buffer.

Declaration: `Property MaxCapacity : Integer`

Visibility: `public`

Access: `Read`

Description: `MaxCapacity` returns the maximum capacity of the string buffer (in characters). It is set when the stringbuilder is created, in the constructor of the builder. The `Capacity` (1833) (and consequently, the `Length` (1832)) will never be allowed to exceed `MaxCapacity`.

See also: `Capacity` (1833), `Length` (1832), `ERangeError` (1820)

76.88 TAnsiStringHelper

76.88.1 Method overview

Page	Method	Description
1836	Compare	
1836	CompareOrdinal	
1836	CompareText	
1839	CompareTo	
1839	Contains	
1836	Copy	
1840	CopyTo	
1840	CountChar	
1837	Create	
1840	DeQuotedString	
1837	EndsText	
1840	EndsWith	
1837	Equals	
1837	Format	
1840	GetHashCode	
1840	IndexOf	
1841	IndexOfAny	
1841	IndexOfAnyUnquoted	
1841	IndexOfUnquoted	
1841	Insert	
1842	IsDelimiter	
1842	IsEmpty	
1837	IsNullOrEmpty	
1837	IsNullOrWhiteSpace	
1838	Join	
1842	LastDelimiter	
1842	LastIndexOf	
1842	LastIndexOfAny	
1838	LowerCase	
1842	PadLeft	
1843	PadRight	
1838	Parse	
1843	QuotedString	
1843	Remove	
1843	Replace	
1843	Split	
1844	StartsWith	
1844	Substring	
1838	ToBoolean	
1845	ToCharArray	
1838	ToDouble	
1839	ToExtended	
1839	ToInt64	
1839	ToInteger	
1845	ToLower	
1845	ToLowerInvariant	
1839	ToSingle	
1845	ToUpper	
1845	ToUpperInvariant	
1845	Trim	
1846	TrimEnd	
1845	TrimLeft	1835
1846	TrimRight	
1846	TrimStart	
1839	UpperCase	

76.88.2 Property overview

Page	Properties	Access	Description
1846	Chars	r	
1846	Length	r	

76.88.3 TAnsiStringHelper.Compare

```

Declaration: class function Compare(const A: AnsiString; const B: AnsiString)
              : Integer; Overload; Static
class function Compare(const A: AnsiString; const B: AnsiString;
                      IgnoreCase: Boolean) : Integer; Overload; Static
class function Compare(const A: AnsiString; const B: AnsiString;
                      Options: TCompareOptions) : Integer; Overload
                      ; Static
class function Compare(const A: AnsiString; IndexA: SizeInt;
                      const B: AnsiString; IndexB: SizeInt;
                      ALen: SizeInt) : Integer; Overload; Static
class function Compare(const A: AnsiString; IndexA: SizeInt;
                      const B: AnsiString; IndexB: SizeInt;
                      ALen: SizeInt; IgnoreCase: Boolean) : Integer
                      ; Overload; Static
class function Compare(const A: AnsiString; IndexA: SizeInt;
                      const B: AnsiString; IndexB: SizeInt;
                      ALen: SizeInt; Options: TCompareOptions) : Integer
                      ; Overload; Static

```

Visibility: public

76.88.4 TAnsiStringHelper.CompareOrdinal

```
Declaration: class function CompareOrdinal(const A: AnsiString; const B: AnsiString)
            : Integer; Overload; Static
            class function CompareOrdinal(const A: AnsiString; IndexA: SizeInt;
            const B: AnsiString; IndexB: SizeInt;
            ALen: SizeInt) : Integer; Overload
            ; Static
```

Visibility: public

76.88.5 TAnsiStringHelper.CompareText

```
Declaration: class function CompareText(const A: AnsiString; const B: AnsiString)
              : Integer; Static
```

Visibility: public

76.88.6 TAnsiStringHelper.Copy

```
Declaration: class function Copy(const Str: AnsiString) : AnsiString; Static
```

Visibility: public

76.88.7 TAnsiStringHelper.Create

```
Declaration: class function Create(AChar: AnsiChar; ACount: SizeInt) : AnsiString
              ; Overload; Static
class function Create(const AValue: Array of AnsiChar) : AnsiString
              ; Overload; Static
class function Create(const AValue: Array of AnsiChar;
              StartIndex: SizeInt; ALen: SizeInt) : AnsiString
              ; Overload; Static
```

Visibility: public

76.88.8 TAnsiStringHelper.EndsText

[illegible]

Visibility: public

76.88.9 TAnsiStringHelper.Equals

```
Declaration: class function Equals(const a: AnsiString; const b: AnsiString)
           : Boolean; Overload; Static
function Equals(const AValue: AnsiString; IgnoreCase: Boolean) : Boolean
           ; Overload
```

Visibility: public

76.88.10 TAnsiStringHelper.Format

```
Declaration: class function Format(const AFormat: AnsiString;
                                const args: Array of const) : AnsiString; Overload
; Static
function Format(const args: Array of const) : AnsiString; Overload
```

Visibility: public

76.88.11 TAnsiStringHelper.IsNullOrEmpty

```
Declaration: class function IsNullOrEmpty(const AValue: AnsiString) : Boolean
                                                    ; Static
```

Visibility: public

76.88.12 TAnsiStringHelper.IsNullOrEmptyWhiteSpace

```
Declaration: class function IsNullOrWhiteSpace(const AValue: AnsiString) : Boolean
                                                    ; Static
```

Visibility: public

76.88.13 TAnsiStringHelper.Join

Declaration: class function Join(const Separator: AnsiString;
 const Values: Array of const) : AnsiString; Overload
 ; Static
 class function Join(const Separator: AnsiString;
 const Values: Array of AnsiString) : AnsiString
 ; Overload; Static
 class function Join(const Separator: AnsiString;
 const Values: Array of AnsiString;
 StartIndex: SizeInt; ACount: SizeInt) : AnsiString
 ; Overload; Static

Visibility: public

76.88.14 TAnsiStringHelper.LowerCase

Declaration: class function LowerCase(const S: AnsiString) : AnsiString; Overload
 ; Static

Visibility: public

76.88.15 TAnsiStringHelper.Parse

Declaration: class function Parse(const AValue: Boolean) : AnsiString; Overload
 ; Static
 class function Parse(const AValue: Extended) : AnsiString; Overload
 ; Static
 class function Parse(const AValue: Int64) : AnsiString; Overload
 ; Static
 class function Parse(const AValue: Integer) : AnsiString; Overload
 ; Static

Visibility: public

76.88.16 TAnsiStringHelper.ToBoolean

Declaration: class function ToBoolean(const S: AnsiString) : Boolean; Overload
 ; Static
 function ToBoolean : Boolean; Overload

Visibility: public

76.88.17 TAnsiStringHelper.ToDouble

Declaration: class function ToDouble(const S: AnsiString) : Double; Overload
 ; Static
 function ToDouble : Double; Overload

Visibility: public

76.88.25 TAnsiStringHelper.CopyTo

Declaration: procedure CopyTo(SourceIndex: SizeInt;
 var destination: Array of AnsiChar;
 DestinationIndex: SizeInt; ACount: SizeInt)

Visibility: public

76.88.26 TAnsiStringHelper.CountChar

Declaration: function CountChar(const C: AnsiChar) : SizeInt

Visibility: public

76.88.27 TAnsiStringHelper.DeQuotedString

Declaration: function DeQuotedString : AnsiString; Overload
 function DeQuotedString(const AQuoteChar: AnsiChar) : AnsiString
 ; Overload

Visibility: public

76.88.28 TAnsiStringHelper.EndsWith

Declaration: function EndsWith(const AValue: AnsiString) : Boolean; Overload
 function EndsWith(const AValue: AnsiString; IgnoreCase: Boolean)
 : Boolean; Overload

Visibility: public

76.88.29 TAnsiStringHelper.GetHashCode

Declaration: function GetHashCode : Integer

Visibility: public

76.88.30 TAnsiStringHelper.IndexOf

Declaration: function IndexOf(AValue: AnsiChar) : SizeInt; Overload
 function IndexOf(const AValue: AnsiString) : SizeInt; Overload
 function IndexOf(AValue: AnsiChar; StartIndex: SizeInt) : SizeInt
 ; Overload
 function IndexOf(const AValue: AnsiString; StartIndex: SizeInt)
 : SizeInt; Overload
 function IndexOf(AValue: AnsiChar; StartIndex: SizeInt; ACount: SizeInt)
 : SizeInt; Overload
 function IndexOf(const AValue: AnsiString; StartIndex: SizeInt;
 ACount: SizeInt) : SizeInt; Overload

Visibility: public

76.88.31 TAnsiStringHelper.IndexOfUnQuoted

Declaration: function IndexOfUnQuoted(const AValue: AnsiString;
 StartQuote: AnsiChar; EndQuote: AnsiChar;
 StartIndex: SizeInt) : SizeInt; Overload

Visibility: public

76.88.32 TAnsiStringHelper.IndexOfAny

Declaration: function IndexOfAny(const AnyOf: Array of AnsiChar) : SizeInt; Overload
 function IndexOfAny(const AnyOf: Array of AnsiChar; StartIndex: SizeInt)
 : SizeInt; Overload
 function IndexOfAny(const AnyOf: Array of AnsiChar;
 StartIndex: SizeInt; ACount: SizeInt) : SizeInt
 ; Overload
 function IndexOfAny(const AnyOf: Array of AnsiString) : SizeInt
 ; Overload
 function IndexOfAny(const AnyOf: Array of AnsiString;
 StartIndex: SizeInt) : SizeInt; Overload
 function IndexOfAny(const AnyOf: Array of AnsiString;
 StartIndex: SizeInt; ACount: SizeInt) : SizeInt
 ; Overload
 function IndexOfAny(const AnyOf: Array of AnsiString;
 StartIndex: SizeInt; ACount: SizeInt;
 out AMatch: SizeInt) : SizeInt; Overload

Visibility: public

76.88.33 TAnsiStringHelper.IndexOfAnyUnquoted

Declaration: function IndexOfAnyUnquoted(const AnyOf: Array of AnsiChar;
 StartQuote: AnsiChar; EndQuote: AnsiChar)
 : SizeInt; Overload
 function IndexOfAnyUnquoted(const AnyOf: Array of AnsiChar;
 StartQuote: AnsiChar; EndQuote: AnsiChar;
 StartIndex: SizeInt) : SizeInt; Overload
 function IndexOfAnyUnquoted(const AnyOf: Array of AnsiChar;
 StartQuote: AnsiChar; EndQuote: AnsiChar;
 StartIndex: SizeInt; ACount: SizeInt)
 : SizeInt; Overload
 function IndexOfAnyUnquoted(const AnyOf: Array of AnsiString;
 StartQuote: AnsiChar; EndQuote: AnsiChar;
 StartIndex: SizeInt; out Matched: SizeInt)
 : SizeInt; Overload

Visibility: public

76.88.34 TAnsiStringHelper.Insert

Declaration: function Insert(StartIndex: SizeInt; const AValue: AnsiString)
 : AnsiString

Visibility: public

76.88.35 TAnsiStringHelper.IsDelimiter

Declaration: function IsDelimiter(const Delimiters: AnsiString; Index: SizeInt)
: Boolean

Visibility: public

76.88.36 TAnsiStringHelper.IsEmpty

Declaration: function IsEmpty : Boolean

Visibility: public

76.88.37 TAnsiStringHelper.LastDelimiter

Declaration: function LastDelimiter(const Delims: AnsiString) : SizeInt

Visibility: public

76.88.38 TAnsiStringHelper.LastIndexOf

Declaration: function LastIndexOf(AValue: AnsiChar) : SizeInt; Overload
function LastIndexOf(const AValue: AnsiString) : SizeInt; Overload
function LastIndexOf(AValue: AnsiChar; AStartIndex: SizeInt) : SizeInt
; Overload
function LastIndexOf(const AValue: AnsiString; AStartIndex: SizeInt)
: SizeInt; Overload
function LastIndexOf(AValue: AnsiChar; AStartIndex: SizeInt;
ACount: SizeInt) : SizeInt; Overload
function LastIndexOf(const AValue: AnsiString; AStartIndex: SizeInt;
ACount: SizeInt) : SizeInt; Overload

Visibility: public

76.88.39 TAnsiStringHelper.LastIndexOfAny

Declaration: function LastIndexOfAny(const AnyOf: Array of AnsiChar) : SizeInt
; Overload
function LastIndexOfAny(const AnyOf: Array of AnsiChar;
AStartIndex: SizeInt) : SizeInt; Overload
function LastIndexOfAny(const AnyOf: Array of AnsiChar;
AStartIndex: SizeInt; ACount: SizeInt) : SizeInt
; Overload

Visibility: public

76.88.40 TAnsiStringHelper.PadLeft

Declaration: function PadLeft(ATotalWidth: SizeInt) : AnsiString; Overload
function PadLeft(ATotalWidth: SizeInt; PaddingChar: AnsiChar)
: AnsiString; Overload

Visibility: public

76.88.41 TAnsiStringHelper.PadRight

```
Declaration: function PadRight(ATotalWidth: SizeInt) : AnsiString; Overload
            function PadRight(ATotalWidth: SizeInt; PaddingChar: AnsiChar)
                        : AnsiString; Overload
```

Visibility: public

76.88.42 TAnsiStringHelper.QuotedString

```
Declaration: function QuotedString : AnsiString; Overload
              function QuotedString(const AQuoteChar: AnsiChar) : AnsiString
                              ; Overload
```

Visibility: public

76.88.43 TAnsiStringHelper.Remove

```
Declaration: function Remove(StartIndex: SizeInt) : AnsiString; Overload
            function Remove(StartIndex: SizeInt; ACount: SizeInt) : AnsiString
                    ; Overload
```

Visibility: public

76.88.44 TAnsiStringHelper.Replace

```
Declaration: function Replace(OldChar: AnsiChar; NewChar: AnsiChar) : AnsiString
                ; Overload
function Replace(OldChar: AnsiChar; NewChar: AnsiChar;
                ReplaceFlags: TReplaceFlags) : AnsiString; Overload
function Replace(const OldValue: AnsiString; const NewValue: AnsiString)
                : AnsiString; Overload
function Replace(const OldValue: AnsiString;
                const NewValue: AnsiString; ReplaceFlags: TReplaceFlags)
                : AnsiString; Overload
```

Visibility: public

76.88.45 TAnsiStringHelper.Split

```

Declaration: function Split(const Separators: Array of AnsiChar) : TAnsiStringArray
                        ; Overload
function Split(const Separators: Array of AnsiChar; ACount: SizeInt)
                : TAnsiStringArray; Overload
function Split(const Separators: Array of AnsiChar;
                Options: TStringSplitOptions) : TAnsiStringArray
                        ; Overload
function Split(const Separators: Array of AnsiChar; ACount: SizeInt;
                Options: TStringSplitOptions) : TAnsiStringArray
                        ; Overload
function Split(const Separators: Array of AnsiString) : TAnsiStringArray
                        ; Overload
function Split(const Separators: Array of AnsiString; ACount: SizeInt)
                : TAnsiStringArray; Overload

```

```

function Split(const Separators: Array of AnsiString;
               Options: TStringSplitOptions) : TAnsiStringArray
    ; Overload
function Split(const Separators: Array of AnsiString; ACount: SizeInt;
               Options: TStringSplitOptions) : TAnsiStringArray
    ; Overload
function Split(const Separators: Array of AnsiChar; AQuote: AnsiChar)
    : TAnsiStringArray; Overload
function Split(const Separators: Array of AnsiChar;
               AQuoteStart: AnsiChar; AQuoteEnd: AnsiChar)
    : TAnsiStringArray; Overload
function Split(const Separators: Array of AnsiChar;
               AQuoteStart: AnsiChar; AQuoteEnd: AnsiChar;
               Options: TStringSplitOptions) : TAnsiStringArray
    ; Overload
function Split(const Separators: Array of AnsiChar;
               AQuoteStart: AnsiChar; AQuoteEnd: AnsiChar;
               ACount: SizeInt) : TAnsiStringArray; Overload
function Split(const Separators: Array of AnsiChar;
               AQuoteStart: AnsiChar; AQuoteEnd: AnsiChar;
               ACount: SizeInt; Options: TStringSplitOptions)
    : TAnsiStringArray; Overload
function Split(const Separators: Array of AnsiString; AQuote: AnsiChar)
    : TAnsiStringArray; Overload
function Split(const Separators: Array of AnsiString;
               AQuoteStart: AnsiChar; AQuoteEnd: AnsiChar)
    : TAnsiStringArray; Overload
function Split(const Separators: Array of AnsiString;
               AQuoteStart: AnsiChar; AQuoteEnd: AnsiChar;
               Options: TStringSplitOptions) : TAnsiStringArray
    ; Overload
function Split(const Separators: Array of AnsiString;
               AQuoteStart: AnsiChar; AQuoteEnd: AnsiChar;
               ACount: SizeInt) : TAnsiStringArray; Overload
function Split(const Separators: Array of AnsiString;
               AQuoteStart: AnsiChar; AQuoteEnd: AnsiChar;
               ACount: SizeInt; Options: TStringSplitOptions)
    : TAnsiStringArray; Overload

```

Visibility: public

76.88.46 TAnsiStringHelper.StartsWith

Declaration: function StartsWith(const AValue: AnsiString) : Boolean; Overload
 function StartsWith(const AValue: AnsiString; IgnoreCase: Boolean)
 : Boolean; Overload

Visibility: public

76.88.47 TAnsiStringHelper.Substring

Declaration: function Substring(AStartIndex: SizeInt) : AnsiString; Overload
 function Substring(AStartIndex: SizeInt; ALen: SizeInt) : AnsiString

; Overload

Visibility: public

76.88.48 TAnsiStringHelper.ToCharArray

Declaration: function ToCharArray : TCharArray; Overload
 function ToCharArray(AStartIndex: SizeInt; ALen: SizeInt) : TCharArray
 ; Overload

Visibility: public

76.88.49 TAnsiStringHelper.ToLower

Declaration: function ToLower : AnsiString; Overload

Visibility: public

76.88.50 TAnsiStringHelper.ToLowerInvariant

Declaration: function ToLowerInvariant : AnsiString

Visibility: public

76.88.51 TAnsiStringHelper.ToUpper

Declaration: function ToUpper : AnsiString; Overload

Visibility: public

76.88.52 TAnsiStringHelper.ToUpperInvariant

Declaration: function ToUpperInvariant : AnsiString

Visibility: public

76.88.53 TAnsiStringHelper.Trim

Declaration: function Trim : AnsiString; Overload
 function Trim(const ATrimChars: Array of AnsiChar) : AnsiString
 ; Overload

Visibility: public

76.88.54 TAnsiStringHelper.TrimLeft

Declaration: function TrimLeft : AnsiString; Overload
 function TrimLeft(const ATrimChars: Array of AnsiChar) : AnsiString
 ; Overload

Visibility: public

76.88.55 TAnsiStringHelper.TrimRight

Declaration: `function TrimRight : AnsiString; Overload`
`function TrimRight(const ATrimChars: Array of AnsiChar) : AnsiString`
`; Overload`

Visibility: public

76.88.56 TAnsiStringHelper.TrimEnd

Declaration: `function TrimEnd(const ATrimChars: Array of AnsiChar) : AnsiString`

Visibility: public

76.88.57 TAnsiStringHelper.TrimStart

Declaration: `function TrimStart(const ATrimChars: Array of AnsiChar) : AnsiString`

Visibility: public

76.88.58 TAnsiStringHelper.Chars

Declaration: `Property Chars[AIndex: SizeInt]: AnsiChar`

Visibility: public

Access: Read

76.88.59 TAnsiStringHelper.Length

Declaration: `Property &Length : SizeInt`

Visibility: public

Access: Read

76.89 TBigEndianUnicodeEncoding**76.89.1 Description**

TBigEndianUnicodeEncoding is the encoding class used to represent the UTF-16 big-endian encoding.

See also: TBigEndianUnicodeEncoding ([1846](#)), TUTF7Encoding ([1982](#)), TMBCSEncoding ([1904](#)), TBigen-
dianUnicodeEncoding ([1846](#))

76.89.2 Method overview

Page	Method	Description
1847	Clone	Clone a TBigEndianUnicodeEncoding instance.
1847	GetPreamble	Return BOM marker bytes.

76.89.3 TBigEndianUnicodeEncoding.Clone

Synopsis: Clone a TBigEndianUnicodeEncoding instance.

Declaration: `function Clone : TEncoding; Override`

Visibility: `public`

Description: `Clone` overrides `TEncoding.Clone` (1872) to provide a clone of the `TBigEndianUnicodeEncoding` instance.

See also: `TEncoding.Clone` (1872)

76.89.4 TBigEndianUnicodeEncoding.GetPreamble

Synopsis: Return BOM marker bytes.

Declaration: `function GetPreamble : TBytes; Override`

Visibility: `public`

Description: `GetPreamble` overrides `TEncoding.GetPreamble` (1874) to return the 2 UTF-16 BOM Marker bytes (\$FF,\$FE).

See also: `TEncoding.GetPreamble` (1874)

76.90 TBooleanHelper

76.90.1 Description

`TBooleanHelper` is a helper type for the `Boolean` type. It contains mostly conversion routines to and from other types.

See also: `TStringHelper` (1609), `TShortIntHelper` (1928), `TSmallIntHelper` (1955), `TWordHelper` (2000), `TCardinalHelper` (1857), `TIntegerHelper` (1893), `TInt64Helper` (1886), `TQWordHelper` (1921), `TNativeIntHelper` (1908), `TByteHelper` (1851), `TByteBoolHelper` (1849), `TWordBoolHelper` (1998), `TLongBoolHelper` (1899)

76.90.2 Method overview

Page	Method	Description
1847	<code>Parse</code>	Convert string value to boolean value.
1848	<code>Size</code>	Return the size (in bytes) of the.
1849	<code>ToInteger</code>	Convert to an integer value.
1848	<code>ToString</code>	Convert a boolean value to string.
1848	<code>TryToParse</code>	Try to convert a string to a boolean value.

76.90.3 TBooleanHelper.Parse

Synopsis: Convert string value to boolean value.

Declaration: `class function Parse(const S: string) : Boolean; Static`

Visibility: `public`

Description: `Parse` attempts to convert the string `S` to a boolean value. It uses the `StrToBool` (1768) function to perform the conversion.

Errors: If `S` does not contain a valid string representation, then an `EConvertError` (1812) exception is raised.

See also: `TBooleanHelper.TryParse` (1848), `TBooleanHelper.ToString` (1848), `TBooleanHelper.ToInteger` (1849)

76.90.4 TBooleanHelper.Size

Synopsis: Return the size (in bytes) of the.

Declaration: `class function Size : Integer; Static`

Visibility: `public`

Description: `Size` returns the size (in bytes) of the boolean value. This is equivalent to `SizeOf(Boolean)`.

See also: `SizeOf` (1546)

76.90.5 TBooleanHelper.ToString

Synopsis: Convert a boolean value to string.

Declaration: `class function ToString(const AValue: Boolean;
UseBoolStrs: TUseBoolStrs) : string; Overload
; Static
function ToString(UseBoolStrs: TUseBoolStrs) : string; Overload`

Visibility: `public`

Description: `ToString` will, in the class method version, convert the `AValue` boolean to a string representation. In the function method version the boolean value itself (`Self`) will be converted.

If the `UseBoolStrs` parameter equals `TUseBoolStrs.True`, then the string representation will use the boolean strings `BoolStrs` (1609). The default value for `UseBoolStrs` is `TUseBoolStrs.False`.

The conversion is done using the `BoolToStr` (1664) function.

See also: `BoolStrs` (1609), `BoolToStr` (1664)

76.90.6 TBooleanHelper.TryParse

Synopsis: Try to convert a string to a boolean value.

Declaration: `class function TryParse(const S: string; out AValue: Boolean)
: Boolean; Static`

Visibility: `public`

Description: `TryParse` will attempt to convert the string `S` to a boolean value. If the attempt is successful, `True` is returned, and the actual value is returned in `AValue`. If the attempt failed, `False` is returned.

See also: `TBooleanHelper.Parse` (1847), `TBooleanHelper.ToString` (1848)

76.90.7 TBooleanHelper.ToInteger

Synopsis: Convert to an integer value.

Declaration: `function ToInteger : Integer`

Visibility: `public`

Description: `ToInteger` will return the boolean value, typecasted to `Integer`.

See also: `TBooleanHelper.ToString` ([1848](#))

76.91 TByteBoolHelper

76.91.1 Description

`TByteBoolHelper` is a helper type for the `ByteBool` type. It contains mostly conversion routines to and from other types.

See also: `TStringHelper` ([1609](#)), `TShortIntHelper` ([1928](#)), `TSmallIntHelper` ([1955](#)), `TWordHelper` ([2000](#)), `TCardinalHelper` ([1857](#)), `TIntegerHelper` ([1893](#)), `TInt64Helper` ([1886](#)), `TQWordHelper` ([1921](#)), `TNativeIntHelper` ([1908](#)), `TByteHelper` ([1851](#)), `TByteBoolHelper` ([1849](#)), `TWordBoolHelper` ([1998](#)), `TLongBoolHelper` ([1899](#))

76.91.2 Method overview

Page	Method	Description
1849	<code>Parse</code>	Convert string value to <code>ByteBool</code> value.
1849	<code>Size</code>	Return the size (in bytes) of the.
1850	<code>ToInteger</code>	Convert to an integer value.
1850	<code>ToString</code>	Convert a <code>ByteBool</code> value to string.
1850	<code>TryToParse</code>	Try to convert a string to a <code>ByteBool</code> value.

76.91.3 TByteBoolHelper.Parse

Synopsis: Convert string value to `ByteBool` value.

Declaration: `class function Parse(const S: string) : Boolean; Static`

Visibility: `public`

Description: `Parse` attempts to convert the string `S` to a `ByteBool` value. It uses the `StrToBool` ([1768](#)) function to perform the conversion.

Errors: If `S` does not contain a valid string representation, then an `EConvertError` ([1812](#)) exception is raised.

See also: `TByteBoolHelper.TryToParse` ([1850](#)), `TByteBoolHelper.ToString` ([1850](#)), `TByteBoolHelper.ToInteger` ([1850](#))

76.91.4 TByteBoolHelper.Size

Synopsis: Return the size (in bytes) of the.

Declaration: `class function Size : Integer; Static`

Visibility: `public`

Description: `Size` returns the size (in bytes) of the `ByteBool` value. This is equivalent to `SizeOf (ByteBool)`.

See also: `SizeOf` ([1546](#))

76.91.5 TByteBoolHelper.ToString

Synopsis: Convert a `ByteBool` value to string.

Declaration:

```
class function ToString(const AValue: Boolean;
                        UseBoolStrs: TUseBoolStrs) : string; Overload
; Static
function ToString(UseBoolStrs: TUseBoolStrs) : string; Overload
```

Visibility: public

Description: `ToString` will, in the class method version, convert the `AValue` `ByteBool` to a string representation. In the function method version the `ByteBool` value itself (`Self`) will be converted.

If the `UseBoolStrs` parameter equals `TUseBoolStrs.True`, then the string representation will use the `ByteBool` strings `BoolStrs` ([1609](#)). The default value for `UseBoolStrs` is `TUseBoolStrs.False`.

The conversion is done using the `BoolToStr` ([1664](#)) function.

See also: `BoolStrs` ([1609](#)), `BoolToStr` ([1664](#))

76.91.6 TByteBoolHelper.TryParse

Synopsis: Try to convert a string to a `ByteBool` value.

Declaration:

```
class function TryParse(const S: string; out AValue: Boolean)
                        : Boolean; Static
```

Visibility: public

Description: `TryParse` will attempt to convert the string `S` to a `ByteBool` value. If the attempt is successful, `True` is returned, and the actual value is returned in `AValue`. If the attempt failed, `False` is returned.

See also: `TByteBoolHelper.Parse` ([1849](#)), `TByteBoolHelper.ToString` ([1850](#))

76.91.7 TByteBoolHelper.ToInteger

Synopsis: Convert to an integer value.

Declaration:

```
function ToInteger : Integer
```

Visibility: public

Description: `ToInteger` will return the `ByteBool` value, typecasted to `Integer`.

See also: `TByteBoolHelper.ToString` ([1850](#))

76.92 TByteHelper

76.92.1 Description

TByteHelper contains some auxiliary routines for a byte-typed ordinal value. It consists mainly of conversion routines to and from other types.

See also: TStringHelper (1609), TShortIntHelper (1928), TSmallIntHelper (1955), TWordHelper (2000), TCardinalHelper (1857), TIntegerHelper (1893), TInt64Helper (1886), TQWordHelper (1921), TNativeIntHelper (1908), TNativeUIntHelper (1915)

76.92.2 Method overview

Page	Method	Description
1855	Clear	Set value to 0.
1854	ClearBit	Set bit to 0.
1855	HighestSetBitPos	Return the position of the leftmost bit set.
1855	LowestSetBitPos	Return the position of the rightmost bit set.
1851	Parse	Convert from a string.
1854	SetBit	Set bit to 1.
1856	SetBitsCount	Count number of bits set.
1852	Size	Size, in bytes, of the byte value.
1855	TestBit	Check bit.
1853	ToBinString	Convert to a binary string representation.
1852	ToBoolean	Convert to a boolean value.
1853	ToDouble	Convert to a double-sized floating point value.
1853	ToExtended	Convert to an extended-sized floating point value.
1854	ToggleBit	Invert bit.
1853	ToHexString	Convert to a hexadecimal string representation.
1854	ToSingle	Convert to a single-sized floating point value.
1852	ToString	Convert the value to string.
1852	TryParse	Try to convert a string to a byte, report success or failure.

76.92.3 Property overview

Page	Properties	Access	Description
1856	Bits	rw	Read or write a bit.
1856	Nibbles	rw	Read or write a nibble.

76.92.4 TByteHelper.Parse

Synopsis: Convert from a string.

Declaration: `class function Parse(const AString: string) : Byte; Static`

Visibility: public

Description: Parse will attempt to convert the string AString to a byte value. It uses the StrToInt (1773) function to perform the conversion, so no localization is taken into account.

Errors: If the string does not contain a valid byte value, an EConvertError (1812) exception is raised.

See also: TByteHelper.ToString (1852), TByteHelper.TryParse (1852), StrToInt (1773)

76.92.5 TByteHelper.Size

Synopsis: Size, in bytes, of the byte value.

Declaration: `class function Size : Integer; Static`

Visibility: public

Description: `Size` returns the size (in bytes) of the byte value. This is equivalent to `SizeOf (Byte)`.

Errors: None.

See also: `SizeOf` ([1546](#))

76.92.6 TByteHelper.ToString

Synopsis: Convert the value to string.

Declaration: `class function ToString(const AValue: Byte) : string; Overload; Static`
`function ToString : string; Overload`

Visibility: public

Description: `ToString` will, in the class function variant of this method, convert `AValue` to a string representation. In the regular method overloaded version of `ToString`, the byte value itself is used. The `IntToStr` ([1738](#)) function is used to do the conversion.

See also: `TByteHelper.Parse` ([1851](#)), `IntToStr` ([1738](#))

76.92.7 TByteHelper.TryParse

Synopsis: Try to convert a string to a byte, report success or failure.

Declaration: `class function TryParse(const AString: string; out AValue: Byte)`
`: Boolean; Static`

Visibility: public

Description: `TryParse` attempts to convert the string `AString` to a byte, and reports the success of the attempt. If the attempt is successful, then `True` is returned, and the actual value of the byte is returned in `AValue`.

It uses the `val` ([1609](#)) function to perform the conversion, so no localization is taken into account.

See also: `TByteHelper.Parse` ([1851](#)), `Val` ([1570](#))

76.92.8 TByteHelper.ToBoolean

Synopsis: Convert to a boolean value.

Declaration: `function ToBoolean : Boolean`

Visibility: public

Description: `ToBoolean` converts the byte value to a boolean: it returns `True` if the value is nonzero, `False` if it is zero.

See also: `TByteHelper.ToSingle` ([1854](#)), `TByteHelper.ToDouble` ([1853](#)), `TByteHelper.ToExtended` ([1853](#)), `TByteHelper.ToString` ([1852](#)), `TByteHelper.ToHexString` ([1853](#))

76.92.9 TByteHelper.ToDouble

Synopsis: Convert to a double-sized floating point value.

Declaration: `function ToDouble : Double`

Visibility: public

Description: `ToDouble` converts the byte value to a double-sized floating point value.

See also: `TByteHelper.ToBoolean` (1852), `TByteHelper.ToExtended` (1853), `TByteHelper.ToSingle` (1854), `TByteHelper.ToString` (1852), `TByteHelper.ToHexString` (1853)

76.92.10 TByteHelper.ToExtended

Synopsis: Convert to an extended-sized floating point value.

Declaration: `function ToExtended : Extended`

Visibility: public

Description: `ToDouble` converts the byte value to an extended-sized floating point value

See also: `TByteHelper.ToBoolean` (1852), `TByteHelper.ToSingle` (1854), `TByteHelper.ToDouble` (1853), `TByteHelper.ToString` (1852), `TByteHelper.ToHexString` (1853)

76.92.11 TByteHelper.ToBinString

Synopsis: Convert to a binary string representation.

Declaration: `function ToBinString : string`

Visibility: public

Description: `ToBinString` converts the byte value to a binary string representation, showing all 8 bits.

See also: `TByteHelper.ToHexString` (1853), `TByteHelper.ToString` (1852)

76.92.12 TByteHelper.ToHexString

Synopsis: Convert to a hexadecimal string representation.

Declaration: `function ToHexString(const AMinDigits: Integer) : string; Overload`
`function ToHexString : string; Overload`

Visibility: public

Description: `ToHexString` converts the byte value to a hexadecimal string representation. The `AMinDigits` argument specifies the minimal number of characters in the resulting string. The string will be left-padded with zeroes if the representation contains less than `AMinDigits` characters.

See also: `TByteHelper.ToBoolean` (1852), `TByteHelper.ToSingle` (1854), `TByteHelper.ToDouble` (1853), `TByteHelper.ToString` (1852), `TByteHelper.ToExtended` (1853)

76.92.13 TByteHelper.ToSingle

Synopsis: Convert to an single-sized floating point value.

Declaration: `function ToSingle : Single`

Visibility: public

Description: `ToSingle` converts the byte value to a single-sized floating point value.

See also: `TByteHelper.ToBoolean` ([1852](#)), `TByteHelper.ToDouble` ([1853](#)), `TByteHelper.ToExtended` ([1853](#)), `TByteHelper.ToString` ([1852](#)), `TByteHelper.HexString` ([1853](#))

76.92.14 TByteHelper.SetBit

Synopsis: Set bit to 1.

Declaration: `function SetBit(const Index: TByteBitIndex) : Byte`

Visibility: public

Description: `SetBit` puts value 1 to bit number determined by argument `Index`.

See also: `TByteHelper.ClearBit` ([1854](#)), `TByteHelper.ToggleBit` ([1854](#)), `TByteHelper.TestBit` ([1855](#)), `TByteHelper.HighestSetBitPos` ([1855](#)), `TByteHelper.LowestSetBitPos` ([1855](#)), `TByteHelper.SetBitsCount` ([1856](#)), `TByteHelper.Bits` ([1856](#))

76.92.15 TByteHelper.ClearBit

Synopsis: Set bit to 0.

Declaration: `function ClearBit(const Index: TByteBitIndex) : Byte`

Visibility: public

Description: `ClearBit` puts value 0 to bit number determined by argument `Index`.

See also: `TByteHelper.SetBit` ([1854](#)), `TByteHelper.ToggleBit` ([1854](#)), `TByteHelper.TestBit` ([1855](#)), `TByteHelper.HighestSetBitPos` ([1855](#)), `TByteHelper.LowestSetBitPos` ([1855](#)), `TByteHelper.SetBitsCount` ([1856](#)), `TByteHelper.Bits` ([1856](#))

76.92.16 TByteHelper.ToggleBit

Synopsis: Invert bit.

Declaration: `function ToggleBit(const Index: TByteBitIndex) : Byte`

Visibility: public

Description: `ToggleBit` reads bit value from bit number determined by argument `Index`, inverts it (if it was 0 it becomes 1, and vice versa), and stores it back.

See also: `TByteHelper.SetBit` ([1854](#)), `TByteHelper.ClearBit` ([1854](#)), `TByteHelper.TestBit` ([1855](#)), `TByteHelper.HighestSetBitPos` ([1855](#)), `TByteHelper.LowestSetBitPos` ([1855](#)), `TByteHelper.SetBitsCount` ([1856](#)), `TByteHelper.Bits` ([1856](#))

76.92.17 TByteHelper.TestBit

Synopsis: Check bit.

Declaration: `function TestBit (const Index: TByteBitIndex) : Boolean`

Visibility: public

Description: `TestBit` reads boolean value (true if bit is 1, and false if bit is 0) from bit number determined by argument `Index`.

See also: `TByteHelper.SetBit` (1854), `TByteHelper.ClearBit` (1854), `TByteHelper.ToggleBit` (1854), `TByteHelper.HighestSetBitPos` (1855), `TByteHelper.LowestSetBitPos` (1855), `TByteHelper.SetBitsCount` (1856), `TByteHelper.Bits` (1856)

76.92.18 TByteHelper.Clear

Synopsis: Set value to 0.

Declaration: `procedure Clear`

Visibility: public

76.92.19 TByteHelper.HighestSetBitPos

Synopsis: Return the position of the leftmost bit set.

Declaration: `function HighestSetBitPos : Int8`

Visibility: public

Description: `HighestSetBitPos` scans the byte, starting at the leftmost position and working towards position 0, and returns the index of the first set bit. The position is measured from the 0-th, rightmost bit. When there are no bits set, the result is -1.

See also: `System.BsrByte` (1609), `TByteHelper.SetBit` (1854), `TByteHelper.ClearBit` (1854), `TByteHelper.ToggleBit` (1854), `TByteHelper.TestBit` (1855), `TByteHelper.LowestSetBitPos` (1855), `TByteHelper.SetBitsCount` (1856), `TByteHelper.Bits` (1856)

76.92.20 TByteHelper.LowestSetBitPos

Synopsis: Return the position of the rightmost bit set.

Declaration: `function LowestSetBitPos : Int8`

Visibility: public

Description: `LowestSetBitPos` scans the byte, starting at position 0 (rightmost position) and returns the index of the first set bit. The position is measured from the 0-th, rightmost bit. When there are no bits set, the result is -1.

See also: `System.BsfByte` (1609), `TByteHelper.SetBit` (1854), `TByteHelper.ClearBit` (1854), `TByteHelper.ToggleBit` (1854), `TByteHelper.TestBit` (1855), `TByteHelper.HighestSetBitPos` (1855), `TByteHelper.SetBitsCount` (1856), `TByteHelper.Bits` (1856)

76.92.21 TByteHelper.SetBitsCount

Synopsis: Count number of bits set.

Declaration: `function SetBitsCount : Byte`

Visibility: `public`

Description: `SetBitsCount` returns the number of bits set in a byte.

See also: `System.PopCnt` ([1609](#)), `TByteHelper.SetBit` ([1854](#)), `TByteHelper.ClearBit` ([1854](#)), `TByteHelper.ToggleBit` ([1854](#)), `TByteHelper.TestBit` ([1855](#)), `TByteHelper.HighestSetBitPos` ([1855](#)), `TByteHelper.LowestSetBitPos` ([1855](#)), `TByteHelper.Bits` ([1856](#))

76.92.22 TByteHelper.Bits

Synopsis: Read or write a bit.

Declaration: `Property Bits[aIndex: TByteBitIndex]: Boolean`

Visibility: `public`

Access: `Read,Write`

Description: `Bits` property reads or writes boolean value to a bit number determined by argument `aIndex`.

```
MyByte.Clear; // %00000000 MyByte equals 0
MyByte.Bits[0] := true; // %00000001 MyByte equals 1
MyByte.Bits[2] := true; // %00000101 MyByte equals 5
WriteLn(MyByte.Bits[2]); // TRUE
```

See also: `TByteHelper.SetBit` ([1854](#)), `TByteHelper.ClearBit` ([1854](#)), `TByteHelper.ToggleBit` ([1854](#)), `TByteHelper.TestBit` ([1855](#)), `TByteHelper.HighestSetBitPos` ([1855](#)), `TByteHelper.LowestSetBitPos` ([1855](#)), `TByteHelper.SetBitsCount` ([1856](#)), `TByteHelper.Nibbles` ([1856](#))

76.92.23 TByteHelper.Nibbles

Synopsis: Read or write a nibble.

Declaration: `Property Nibbles[aIndex: TByteNibbleIndex]: nibble`

Visibility: `public`

Access: `Read,Write`

Description: `Nibbles` property reads or writes nibble value to a nibble number determined by argument `aIndex`.

```
MyByte.Clear; // %00000000 MyByte equals 0
MyByte.Nibbles[1] := 3; // %00110000 MyByte equals 48
WriteLn(MyByte.Nibbles[1]); // 3
```

See also: `TByteHelper.Bits` ([1856](#))

76.93 TCardinalHelper

76.93.1 Description

TCardinalHelper contains some auxiliary routines for a Cardinal-typed ordinal value. It consists mainly of conversion routines to and from other types.

See also: TStringHelper ([1609](#)), TShortIntHelper ([1928](#)), TSmallIntHelper ([1955](#)), TWordHelper ([2000](#)), TByteHelper ([1851](#)), TIntegerHelper ([1893](#)), TInt64Helper ([1886](#)), TQWordHelper ([1921](#)), TNativeIntHelper ([1908](#)), TNativeUIntHelper ([1915](#))

76.93.2 Method overview

Page	Method	Description
1861	Clear	Set value to 0.
1860	ClearBit	Set bit to 0.
1861	HighestSetBitPos	Return the position of the leftmost bit set.
1861	LowestSetBitPos	Return the position of the rightmost bit set.
1857	Parse	Convert from a string.
1860	SetBit	Set bit to 1.
1862	SetBitsCount	Count number of bits set.
1858	Size	Size, in bytes, of the Cardinal value.
1861	TestBit	Check bit.
1859	ToBinString	Convert to a binary string representation.
1858	ToBoolean	Convert to a boolean value.
1859	ToDouble	Convert to a double-sized floating point value.
1859	ToExtended	Convert to an extended-sized floating point value.
1860	ToggleBit	Invert bit.
1859	ToHexString	Convert to a hexadecimal string representation.
1860	ToSingle	Convert to an single-sized floating point value.
1858	ToString	Convert the value to string.
1858	TryParse	Try to convert a string to a Cardinal, report success or failure.

76.93.3 Property overview

Page	Properties	Access	Description
1862	Bits	rw	Read or write a bit.
1863	Bytes	rw	Read or write a byte.
1862	Nibbles	rw	Read or write a nibble.
1863	Words	rw	Read or write a word.

76.93.4 TCardinalHelper.Parse

Synopsis: Convert from a string.

Declaration: `class function Parse(const AString: string) : Cardinal; Static`

Visibility: public

Description: Parse will attempt to convert the string AString to a Cardinal value. It uses the StrToInt ([1773](#)) function to perform the conversion, so no localization is taken into account.

Errors: If the string does not contain a valid Cardinal value, an EConvertError ([1812](#)) exception is raised.

See also: TCardinalHelper.ToString ([1858](#)), TCardinalHelper.TryParse ([1858](#)), StrToInt ([1773](#))

76.93.5 TCardinalHelper.Size

Synopsis: Size, in bytes, of the Cardinal value.

Declaration: `class function Size : Integer; Static`

Visibility: public

Description: `Size` returns the size (in Cardinals) of the Cardinal value. This is equivalent to `SizeOf (Cardinal)`.

Errors: None.

See also: `SizeOf` ([1546](#))

76.93.6 TCardinalHelper.ToString

Synopsis: Convert the value to string.

Declaration: `class function ToString(const AValue: Cardinal) : string; Overload
; Static
function ToString : string; Overload`

Visibility: public

Description: `ToString` will, in the class function variant of this method, convert `AValue` to a string representation. In the regular method overloaded version of `ToString`, the Cardinal value itself is used. The `IntToStr` ([1738](#)) function is used to do the conversion.

See also: `TCardinalHelper.Parse` ([1857](#)), `IntToStr` ([1738](#))

76.93.7 TCardinalHelper.TryParse

Synopsis: Try to convert a string to a Cardinal, report success or failure.

Declaration: `class function TryParse(const AString: string; out AValue: Cardinal)
: Boolean; Static`

Visibility: public

Description: `TryParse` attempts to convert the string `AString` to a Cardinal, and reports the success of the attempt. If the attempt is successful, then `True` is returned, and the actual value of the Cardinal is returned in `AValue`.

It uses the `val` ([1609](#)) function to perform the conversion, so no localization is taken into account.

See also: `TCardinalHelper.Parse` ([1857](#)), `Val` ([1570](#))

76.93.8 TCardinalHelper.ToBoolean

Synopsis: Convert to a boolean value.

Declaration: `function ToBoolean : Boolean`

Visibility: public

Description: `ToBoolean` converts the Cardinal value to a boolean: it returns `True` if the value is nonzero, `False` if it is zero.

See also: `TCardinalHelper.ToSingle` ([1860](#)), `TCardinalHelper.ToDouble` ([1859](#)), `TCardinalHelper.ToExtended` ([1859](#)), `TCardinalHelper.ToString` ([1858](#)), `TCardinalHelper.ToHexString` ([1859](#))

76.93.9 TCardinalHelper.ToDouble

Synopsis: Convert to a double-sized floating point value.

Declaration: `function ToDouble : Double`

Visibility: `public`

Description: `ToDouble` converts the Cardinal value to a double-sized floating point value.

See also: `TCardinalHelper.ToBoolean` (1858), `TCardinalHelper.ToExtended` (1859), `TCardinalHelper.ToSingle` (1860), `TCardinalHelper.ToString` (1858), `TCardinalHelper.ToHexString` (1859)

76.93.10 TCardinalHelper.ToExtended

Synopsis: Convert to an extended-sized floating point value.

Declaration: `function ToExtended : Extended`

Visibility: `public`

Description: `ToDouble` converts the Cardinal value to an extended-sized floating point value.

See also: `TCardinalHelper.ToBoolean` (1858), `TCardinalHelper.ToSingle` (1860), `TCardinalHelper.ToDouble` (1859), `TCardinalHelper.ToString` (1858), `TCardinalHelper.ToHexString` (1859)

76.93.11 TCardinalHelper.ToBinString

Synopsis: Convert to a binary string representation.

Declaration: `function ToBinString : string`

Visibility: `public`

Description: `ToBinString` converts the Cardinal value to a binary string representation, showing all 32 bits.

See also: `TCardinalHelper.ToHexString` (1859), `TCardinalHelper.ToString` (1858)

76.93.12 TCardinalHelper.ToHexString

Synopsis: Convert to a hexadecimal string representation.

Declaration: `function ToHexString(const AMinDigits: Integer) : string; Overload`
`function ToHexString : string; Overload`

Visibility: `public`

Description: `ToHexString` converts the Cardinal value to a hexadecimal string representation. The `AMinDigits` argument specifies the minimal number of characters in the resulting string. The string will be left-padded with zeroes if the representation contains less than `AMinDigits` characters.

See also: `TCardinalHelper.ToBoolean` (1858), `TCardinalHelper.ToSingle` (1860), `TCardinalHelper.ToDouble` (1859), `TCardinalHelper.ToString` (1858), `TCardinalHelper.ToExtended` (1859)

76.93.13 TCardinalHelper.ToSingle

Synopsis: Convert to an single-sized floating point value.

Declaration: `function ToSingle : Single`

Visibility: public

Description: `ToSingle` converts the Cardinal value to a single-sized floating point value.

See also: `TCardinalHelper.ToBoolean` (1858), `TCardinalHelper.ToDouble` (1859), `TCardinalHelper.ToExtended` (1859), `TCardinalHelper.ToString` (1858), `TCardinalHelper.ToHexString` (1859)

76.93.14 TCardinalHelper.SetBit

Synopsis: Set bit to 1.

Declaration: `function SetBit(const Index: TCardinalBitIndex) : Cardinal`

Visibility: public

Description: `SetBit` puts value 1 to bit number determined by argument `Index`.

See also: `TCardinalHelper.ClearBit` (1860), `TCardinalHelper.ToggleBit` (1860), `TCardinalHelper.TestBit` (1861), `TCardinalHelper.HighestSetBitPos` (1861), `TCardinalHelper.LowestSetBitPos` (1861), `TCardinalHelper.SetBitsCount` (1862), `TCardinalHelper.Bits` (1862)

76.93.15 TCardinalHelper.ClearBit

Synopsis: Set bit to 0.

Declaration: `function ClearBit(const Index: TCardinalBitIndex) : Cardinal`

Visibility: public

Description: `ClearBit` puts value 0 to bit number determined by argument `Index`.

See also: `TCardinalHelper.SetBit` (1860), `TCardinalHelper.ToggleBit` (1860), `TCardinalHelper.TestBit` (1861), `TCardinalHelper.HighestSetBitPos` (1861), `TCardinalHelper.LowestSetBitPos` (1861), `TCardinalHelper.SetBitsCount` (1862), `TCardinalHelper.Bits` (1862)

76.93.16 TCardinalHelper.ToggleBit

Synopsis: Invert bit.

Declaration: `function ToggleBit(const Index: TCardinalBitIndex) : Cardinal`

Visibility: public

Description: `ToggleBit` reads bit value from bit number determined by argument `Index`, inverts it (if it was 0 it becomes 1, and vice versa), and stores it back.

See also: `TCardinalHelper.SetBit` (1860), `TCardinalHelper.ToggleBit` (1860), `TCardinalHelper.TestBit` (1861), `TCardinalHelper.HighestSetBitPos` (1861), `TCardinalHelper.LowestSetBitPos` (1861), `TCardinalHelper.SetBitsCount` (1862), `TCardinalHelper.Bits` (1862)

76.93.17 TCardinalHelper.TestBit

Synopsis: Check bit.

Declaration: `function TestBit (const Index: TCardinalBitIndex) : Boolean`

Visibility: public

Description: `TestBit` reads boolean value (true if bit is 1, and false if bit is 0) from bit number determined by argument `Index`.

See also: `TCardinalHelper.SetBit` (1860), `TCardinalHelper.ToggleBit` (1860), `TCardinalHelper.TestBit` (1861), `TCardinalHelper.HighestSetBitPos` (1861), `TCardinalHelper.LowestSetBitPos` (1861), `TCardinalHelper.SetBitsCount` (1862), `TCardinalHelper.Bits` (1862)

76.93.18 TCardinalHelper.Clear

Synopsis: Set value to 0.

Declaration: `procedure Clear`

Visibility: public

76.93.19 TCardinalHelper.HighestSetBitPos

Synopsis: Return the position of the leftmost bit set.

Declaration: `function HighestSetBitPos : Int8`

Visibility: public

Description: `HighestSetBitPos` scans the Cardinal, starting at the leftmost position and working towards position 0, and returns the index of the first set bit. The position is measured from the 0-th, rightmost bit. When there are no bits set, the result is -1.

See also: `System.BsrByte` (1609), `TCardinalHelper.SetBit` (1860), `TCardinalHelper.ClearBit` (1860), `TCardinalHelper.ToggleBit` (1860), `TCardinalHelper.TestBit` (1861), `TCardinalHelper.LowestSetBitPos` (1861), `TCardinalHelper.SetBitsCount` (1862), `TCardinalHelper.Bits` (1862)

76.93.20 TCardinalHelper.LowestSetBitPos

Synopsis: Return the position of the rightmost bit set.

Declaration: `function LowestSetBitPos : Int8`

Visibility: public

Description: `LowestSetBitPos` scans the Cardinal, starting at position 0 (rightmost position) and returns the index of the first set bit. The position is measured from the 0-th, rightmost bit. When there are no bits set, the result is -1.

See also: `System.BsfByte` (1609), `TCardinalHelper.SetBit` (1860), `TCardinalHelper.ClearBit` (1860), `TCardinalHelper.ToggleBit` (1860), `TCardinalHelper.TestBit` (1861), `TCardinalHelper.HighestSetBitPos` (1861), `TCardinalHelper.SetBitsCount` (1862), `TCardinalHelper.Bits` (1862)

76.93.21 TCardinalHelper.SetBitsCount

Synopsis: Count number of bits set.

Declaration: `function SetBitsCount : Byte`

Visibility: `public`

Description: `SetBitsCount` returns the number of bits set in a Cardinal.

See also: `System.PopCnt` (1609), `TCardinalHelper.SetBit` (1860), `TCardinalHelper.ClearBit` (1860), `TCardinalHelper.ToggleBit` (1860), `TCardinalHelper.TestBit` (1861), `TCardinalHelper.HighestSetBitPos` (1861), `TCardinalHelper.LowestSetBitPos` (1861), `TCardinalHelper.Bits` (1862)

76.93.22 TCardinalHelper.Bits

Synopsis: Read or write a bit.

Declaration: `Property Bits[aIndex: TCardinalBitIndex]: Boolean`

Visibility: `public`

Access: `Read,Write`

Description: `Bits` property reads or writes boolean value to a bit number determined by argument `aIndex`.

```
MyCardinal.Clear; // %00000000000000000000000000000000 MyCardinal equals 0
MyCardinal.Bits[0] := true; // %00000000000000000000000000000001 MyCardinal equals 1
MyCardinal.Bits[2] := true; // %00000000000000000000000000000101 MyCardinal equals 5
WriteLn(MyCardinal.Bits[2]); // TRUE
```

See also: `TCardinalHelper.SetBit` (1860), `TCardinalHelper.ClearBit` (1860), `TCardinalHelper.ToggleBit` (1860), `TCardinalHelper.TestBit` (1861), `TCardinalHelper.HighestSetBitPos` (1861), `TCardinalHelper.LowestSetBitPos` (1861), `TCardinalHelper.SetBitsCount` (1862), `TCardinalHelper.Nibbles` (1862), `TCardinalHelper.Bytes` (1863), `TCardinalHelper.Words` (1863)

76.93.23 TCardinalHelper.Nibbles

Synopsis: Read or write a nibble.

Declaration: `Property Nibbles[aIndex: TCardinalNibbleIndex]: nibble`

Visibility: `public`

Access: `Read,Write`

Description: `Nibbles` property reads or writes nibble value to a nibble number determined by argument `aIndex`.

```
MyCardinal.Clear; // %00000000000000000000000000000000 MyCardinal equals 0
MyCardinal.Nibbles[1] := 3; // %0000000000000000000000000110000 MyCardinal equals 48
WriteLn(MyCardinal.Nibbles[1]); // 3
```

See also: `TCardinalHelper.Bits` (1862), `TCardinalHelper.Bytes` (1863), `TCardinalHelper.Words` (1863)

76.93.24 TCardinalHelper.Bytes

Synopsis: Read or write a byte.

Declaration: Property Bytes[aIndex: TCardinalByteIndex]: Byte

Visibility: public

Access: Read,Write

Description: Bytes property reads or writes byte value to a byte number determined by argument aIndex.

```
MyCardinal.Clear; // %00000000000000000000000000000000 MyCardinal equals 0
MyCardinal.Bytes[1] := 1; // %0000000000000000000000000100000000 MyCardinal equals 2
WriteLn(MyCardinal.Bytes[1]); // 1
```

See also: TCardinalHelper.Bits ([1862](#)), TCardinalHelper.Nibbles ([1862](#)), TCardinalHelper.Words ([1863](#))

76.93.25 TCardinalHelper.Words

Synopsis: Read or write a word.

Declaration: Property Words[aIndex: TCardinalWordIndex]: Word

Visibility: public

Access: Read,Write

Description: Words property reads or writes word value to a word number determined by argument aIndex.

```
MyCardinal.Clear; // %00000000000000000000000000000000 MyCardinal equals 0
MyCardinal.Words[1] := 1; // %00000000000000000100000000000000 MyCardinal equals 655
WriteLn(MyCardinal.Words[1]); // 1
```

See also: TCardinalHelper.Bits ([1862](#)), TCardinalHelper.Nibbles ([1862](#)), TCardinalHelper.Bytes ([1863](#))

76.94 TCurrencyHelper

76.94.1 Method overview

Page	Method	Description
1864	Ceil	
1864	Floor	
1864	Frac	
1864	Parse	
1864	Size	
1864	ToString	
1864	Trunc	
1865	TryParse	

76.94.2 Property overview

Page	Properties	Access	Description
1865	MaxValue	r	
1865	MinValue	r	

76.94.3 TCurrencyHelper.Ceil

Declaration: function Ceil : Int64

Visibility: public

76.94.4 TCurrencyHelper.Floor

Declaration: function Floor : Int64

Visibility: public

76.94.5 TCurrencyHelper.Frac

Declaration: function Frac : Currency

Visibility: public

76.94.6 TCurrencyHelper.Parse

Declaration: class function Parse(const S: string;
 const AFormatSettings: TFormatSettings) : Currency
 ; Overload; Static
 class function Parse(const S: string) : Currency; Overload; Static

Visibility: public

76.94.7 TCurrencyHelper.Size

Declaration: class function Size : Integer; Static

Visibility: public

76.94.8 TCurrencyHelper.ToString

Declaration: function ToString(const AFormatSettings: TFormatSettings) : string
 ; Overload
 function ToString : string; Overload
 class function ToString(const Value: Currency;
 const AFormatSettings: TFormatSettings) : string
 ; Overload; Static
 class function ToString(const Value: Currency) : string; Overload
 ; Static

Visibility: public

76.94.9 TCurrencyHelper.Trunc

Declaration: function Trunc : Int64

Visibility: public

76.94.10 TCurrencyHelper.TryParse

Declaration: `class function TryParse(const S: string; out Value: Currency;
 const AFormatSettings: TFormatSettings) : Boolean
 ; Overload; Static
 class function TryParse(const S: string; out Value: Currency) : Boolean
 ; Overload; Static`

Visibility: public

76.94.11 TCurrencyHelper.MaxValue

Declaration: `Property MaxValue : Currency`

Visibility: public

Access: Read

76.94.12 TCurrencyHelper.MinValue

Declaration: `Property MinValue : Currency`

Visibility: public

Access: Read

76.95 TDoubleHelper**76.95.1 Description**

`TDoubleHelper` is the helper type for the Double-sized floating point type. It contains some conversion methods, as well as access to the low-level structure of the floating-point representation of a Double.

See also: `TDoubleHelper` ([1865](#)), `TExtendedHelper` ([1878](#))

76.95.2 Method overview

Page	Method	Description
1868	<code>BuildUp</code>	Build a Double-sized floating point from its composing parts.
1869	<code>Exponent</code>	Exponent of the floating-point value.
1869	<code>Fraction</code>	Fraction of the floating-point value.
1866	<code>IsInfinity</code>	Check whether a value is positive or negative infinity.
1866	<code>IsNan</code>	Check whether a value equals NaN.
1866	<code>IsNegativeInfinity</code>	Check whether a value is negative infinity.
1867	<code>IsPositiveInfinity</code>	Check whether a value is positive infinity.
1869	<code>Mantissa</code>	Mantissa of the floating-point.
1867	<code>Parse</code>	Convert a string to a floating point value.
1867	<code>Size</code>	Size (in bytes) of a Double-sized floating point value.
1870	<code>SpecialType</code>	Return the type of the Double-sized floating point value.
1867	<code>ToString</code>	Convert a Double-sized floating point value to a string.
1868	<code>TryParse</code>	Try to convert a string to a Double-sized floating point value.

76.95.3 Property overview

Page	Properties	Access	Description
1870	Bytes	rw	Indexed access to the individual bytes of the floating point value.
1871	Exp	rw	The bit pattern of the exponent as stored in memory.
1871	Frac	rw	Bitpattern that makes up the fractional part.
1870	Sign	rw	Sign of the floating point value.
1870	Words	rw	Indexed access to the words that make up the floating point value.

76.95.4 TDoubleHelper.IsInfinity

Synopsis: Check whether a value is positive or negative infinity.

```
Declaration: class function IsInfinity(const AValue: Double) : Boolean;  Overload
                ;  Static
                function IsInfinity : Boolean;  Overload
```

Visibility: public

Description: `IsInfinity` checks whether a Double-sized floating point value represents a positive or negative infinity. If so, it returns `True`. When the class function version is used, the value can be specified using `AValue`. In the method version, the used value is `(Self)`.

See also: [TDoubleHelper.IsNan \(1866\)](#), [TDoubleHelper.IsPositiveInfinity \(1867\)](#), [TDoubleHelper.IsNegativeInfinity \(1866\)](#)

76.95.5 TDoubleHelper.IsNan

Synopsis: Check whether a value equals NaN.

```
Declaration: class function IsNan(const AValue: Double) : Boolean; Overload; Static
            function IsNan : Boolean; Overload
```

Visibility: public

Description: `IsNaN` checks whether a Double-sized floating point value is NaN (Not a Number). If so, it returns `True`. When the class function version is used, the value can be specified using `AValue`. In the method version, the used value is `(Self)`.

See also: [TDoubleHelper.IsInfinity \(1866\)](#), [TDoubleHelper.IsPositiveInfinity \(1867\)](#), [TDoubleHelper.IsNegativeInfinity \(1866\)](#)

76.95.6 TDoubleHelper.IsNegativeInfinity

Synopsis: Check whether a value is negative infinity.

```
Declaration: class function IsNegativeInfinity(const AValue: Double) : Boolean
                                     ; Overload; Static
      function IsNegativeInfinity : Boolean; Overload
```

Visibility: public

Description: `IsNegativeInfinity` checks whether a Double-sized floating point value represents a negative infinity. If so, it returns `True`. When the class function version is used, the value can be specified using `AValue`. In the method version, the used value is `(Self)`.

See also: `TDoubeHelper.IsNan` ([1866](#)), `TDoubeHelper.IsPositiveInfinity` ([1867](#)), `TDoubeHelper.IsInfinity` ([1866](#))

76.95.7 TDoubleHelper.IsPositiveInfinity

Synopsis: Check whether a value is positive infinity.

Declaration: `class function IsPositiveInfinity(const AValue: Double) : Boolean
; Overload; Static
function IsPositiveInfinity : Boolean; Overload`

Visibility: public

Description: `IsPositiveInfinity` checks whether a Double-sized floating point value represents a positive infinity. If so, it returns `True`. When the class function version is used, the value can be specified using `AValue`. In the method version, the used value is `(Self)`.

See also: `TDoubleHelper.IsNan` (1866), `TDoubleHelper.IsNegativeInfinity` (1866), `TDoubleHelper.IsInfinity` (1866)

76.95.8 TDoubleHelper.Parse

Synopsis: Convert a string to a floating point value.

Declaration: `class function Parse(const AString: string) : Double; Overload; Static
class function Parse(const AString: string;
const AFormatSettings: TFormatSettings) : Double
; Overload; Static`

Visibility: public

Description: `Parse` will try to convert `AString` to a Double-sized floating point value. It will take into account internationalization settings. (it uses `FloatToStr`).

Errors: If the string `AString` is not a valid floating-point value, a `EConvertError` (1812) exception is raised.

See also: `FloatToStr` (1709), `TDoubleHelper.ToString` (1867), `TDoubleHelper.TryParse` (1868)

76.95.9 TDoubleHelper.Size

Synopsis: Size (in bytes) of a Double-sized floating point value.

Declaration: `class function Size : Integer; Static`

Visibility: public

Description: `Size` is the size (in bytes) of a Double-sized floating point value. It is equivalent to calling `SizeOf(Double)`.

See also: `SizeOf` (1546)

76.95.10 TDoubleHelper.ToString

Synopsis: Convert a Double-sized floating point value to a string.

Declaration: `class function ToString(const AValue: Double) : string; Overload
; Static
class function ToString(const AValue: Double;
const AFormat: TFloatFormat;
const APrecision: Integer;
const ADigits: Integer) : string; Overload`

```

; Static
class function ToString(const AValue: Double;
    const AFormat: TFloatFormat;
    const APrecision: Integer;
    const ADigits: Integer;
    const AFormatSettings: TFormatSettings) : string
; Overload; Static
class function ToString(const AValue: Double;
    const AFormatSettings: TFormatSettings) : string
; Overload; Static
function ToString(const AFormat: TFloatFormat;
    const APrecision: Integer; const ADigits: Integer)
    : string; Overload
function ToString(const AFormat: TFloatFormat;
    const APrecision: Integer; const ADigits: Integer;
    const AFormatSettings: TFormatSettings) : string
; Overload
function ToString(const AFormatSettings: TFormatSettings) : string
; Overload
function ToString : string; Overload

```

Visibility: public

Description: `ToString` will convert `AValue` (or `Self` in the plain method version) to a string. Optionally `FormatSettings` can be specified, to be able to specify the decimal separator character to use.

Additionally, a precision `APrecision` and number of digits `ADigits` can be specified, in conjunction with a `AFormat` parameter to specify the form in which the floating-point value must be represented. (see `TFloatFormat` (1632) for an explanation of the various values). In this case, `FloatToStrF` (1710) is used to format the value. In the absence of these parameters, `FloatToStr` (1709) is called.

See also: `FloatToStr` (1709), `FloatToStrF` (1710), `TFloatFormat` (1632)

76.95.11 TDoubleHelper.TryParse

Synopsis: Try to convert a string to a Double-sized floating point value.

Declaration:

```

class function TryParse(const AString: string; out AValue: Double)
    : Boolean; Overload; Static
class function TryParse(const AString: string; out AValue: Double;
    const AFormatSettings: TFormatSettings) : Boolean
; Overload; Static

```

Visibility: public

Description: `TryParse` attempts to convert the string `AString` to a Double-sized floating point value and reports `True` if the conversion was successful. In that case the parsed value is returned in `AValue`.

If the conversion failed, `False` is returned.

See also: `TDoubleHelper.Parse` (1867), `TDoubleHelper.ToString` (1867)

76.95.12 TDoubleHelper.BuildUp

Synopsis: Build a Double-sized floating point from its composing parts.

Declaration: `procedure BuildUp(const ASignFlag: Boolean; const AMantissa: QWord;
const AExponent: Integer)`

Visibility: public

Description: `BuildUp` will compose a Double-sized floating point value from the sign `ASignFlag`, mantissa `AMantissa` and exponent `AExponent`. It simply sets the `Sign` (1870), `Exp` (1871) and `Frac` (1871) properties in 1 call.

See also: `TDoubleHelper.Sign` (1870), `TDoubleHelper.Exp` (1871), `TDoubleHelper.Frac` (1871)

76.95.13 TDoubleHelper.Exponent

Synopsis: Exponent of the floating-point value.

Declaration: `function Exponent : Integer`

Visibility: public

Description: `Exponent` is the value X in the representation of the floating-point value in $m \cdot 2^X$, i.e. the exponent.

See also: `TDoubleHelper.Sign` (1870), `TDoubleHelper.Exp` (1871), `TDoubleHelper.Frac` (1871), `TDoubleHelper.Fraction` (1869), `TDoubleHelper.Mantissa` (1869)

76.95.14 TDoubleHelper.Fraction

Synopsis: Fraction of the floating-point value.

Declaration: `function Fraction : Extended`

Visibility: public

Description: `Fraction` is the decimal part of the floating-point value.

See also: `TDoubleHelper.Sign` (1870), `TDoubleHelper.Exp` (1871), `TDoubleHelper.Exponent` (1869), `TDoubleHelper.Frac` (1871), `TDoubleHelper.Mantissa` (1869)

76.95.15 TDoubleHelper.Mantissa

Synopsis: Mantissa of the floating-point.

Declaration: `function Mantissa : QWord`

Visibility: public

Description: `Mantissa` is the value of the significand without the hidden bit. This means it the plain bit pattern as it is stored in memory.

See also: `TDoubleHelper.Sign` (1870), `TDoubleHelper.Exp` (1871), `TDoubleHelper.Exponent` (1869), `TDoubleHelper.Frac` (1871), `TDoubleHelper.Fraction` (1869)

76.95.16 TDoubleHelper.SpecialType

Synopsis: Return the type of the Double-sized floating point value.

Declaration: `function SpecialType : TFloatSpecial`

Visibility: `public`

Description: `SpecialType` checks whether the Double-sized floating point value equals one of several special values, and returns an enumerated value describing which value this is. See `TFloatSpecial` (1609) for an explanation of the possible values.

See also: `TFloatSpecial` (1609)

76.95.17 TDoubleHelper.Bytes

Synopsis: Indexed access to the individual bytes of the floating point value.

Declaration: `Property Bytes[AIndex: Cardinal]: Byte`

Visibility: `public`

Access: `Read,Write`

Description: `Bytes` can be used to get or set the various bytes that make up the Double-sized floating point value. The index runs from 0 to `Size-1`.

See also: `TDoubleHelper.Words` (1870), `TDoubleHelper.Size` (1867)

76.95.18 TDoubleHelper.Words

Synopsis: Indexed access to the words that make up the floating point value.

Declaration: `Property Words[AIndex: Cardinal]: Word`

Visibility: `public`

Access: `Read,Write`

Description: `Words` can be used to get or set the various bytes that make up the Double-sized floating point value. The index runs from 0 to $(\text{Size}-1) \div 2$.

See also: `TDoubleHelper.Bytes` (1870), `TDoubleHelper.Size` (1867)

76.95.19 TDoubleHelper.Sign

Synopsis: Sign of the floating point value.

Declaration: `Property Sign : Boolean`

Visibility: `public`

Access: `Read,Write`

Description: `Sign` returns `True` if the sign bit of the value is set (i.e. it is a negative value) or `False` if it is not set (i.e. it is a positive value).

See also: `TDoubleHelper.Bytes` (1870), `TDoubleHelper.Exp` (1871), `TDoubleHelper.Frac` (1871), `TDoubleHelper.Mantissa` (1869), `TDoubleHelper.Fraction` (1869), `TDoubleHelper.Exponent` (1869)

76.95.20 TDoubleHelper.Exp

Synopsis: The bit pattern of the exponent as stored in memory.

Declaration: `Property Exp : QWord`

Visibility: public

Access: Read,Write

Description: `Exp` is the internal representation of the Exponent (1869).

See also: `TDoubleHelper.Bytes` (1870), `TDoubleHelper.Sign` (1870), `TDoubleHelper.Frac` (1871), `TDoubleHelper.Mantissa` (1869), `TDoubleHelper.Fraction` (1869), `TDoubleHelper.Exponent` (1869)

76.95.21 TDoubleHelper.Frac

Synopsis: Bitpattern that makes up the fractional part.

Declaration: `Property Frac : QWord`

Visibility: public

Access: Read,Write

Description: `Frac` is the bit pattern representing the fractional part (significand) including the preceding 1 (the hidden bit).

See also: `TDoubleHelper.Bytes` (1870), `TDoubleHelper.Sign` (1870), `TDoubleHelper.Exp` (1871), `TDoubleHelper.Mantissa` (1869), `TDoubleHelper.Fraction` (1869), `TDoubleHelper.Exponent` (1869)

76.96 TEncoding

76.96.1 Description

`TEncoding` is a mostly abstract class that contains various methods to deal with different encodings in single- and double-byte strings. In practice, one instance of a descendent of this class for each needed encoding can be instantiated and used. The class contains several class methods and properties to create such instances, and has several often-used instances available as class properties, in particular the default encoding.

When using encodings (and in particular `TEncoding.Default` (1876) or `TEncoding.ANSI` (1876)), if the `DefaultSystemCodePage` (1415) changes, you should call `TEncoding.FreeEncodings` (1871) to regenerate the encodings.

See also: `TEncoding.ANSI` (1876), `TEncoding.ASCII` (1876), `TEncoding.Default` (1876), `TEncoding.BigEndianUnicode` (1876), `TEncoding.Unicode` (1877), `TEncoding.UTF7` (1877), `TEncoding.UTF8` (1878)

76.96.2 Method overview

Page	Method	Description
1872	Clone	Clone a TEncoding instance.
1872	Convert	Convert an array of bytes from one encoding to another.
1873	GetBufferEncoding	Attempt to guess the encoding of a buffer.
1873	GetEncoding	Get an encoding instance for a given codepage.
1874	GetMaxByteCount	Returns the maximum number of bytes needed to represent a string.
1874	GetMaxCharCount	Return the maximum number of characters that can be represented in a number of bytes.
1874	GetPreamble	Return the BOM Marker used by the encoding.
1875	GetString	Return a string based on an array of bytes.
1873	IsStandardEncoding	Check if the encoding is one of the standard encodings.

76.96.3 Property overview

Page	Properties	Access	Description
1876	ANSI	r	Ansi encoding instance.
1876	ASCII	r	ASCII encoding instance.
1876	BigEndianUnicode	r	Big-endian Unicode (UTF16BE) encoding instance.
1875	CodePage	r	Codepage for this encoding.
1876	Default	r	Default codepage.
1875	EncodingName	r	Name of this encoding.
1875	IsSingleByte	r	Is the encoding a single-byte encoding or not ?
1877	SystemEncoding	r	System encoding.
1877	Unicode	r	UTF16 encoding instance.
1877	UTF7	r	UTF7 encoding instance.
1878	UTF8	r	UTF8 encoding instance.

76.96.4 TEncoding.Clone

Synopsis: Clone a TEncoding instance.

Declaration: `function Clone : TEncoding; Virtual`

Visibility: public

Description: Clone creates a copy of a TEncoding instance. This method returns Nil in TEncoding and must be implemented in descendent classes.

See also: TEncoding.Create ([1871](#))

76.96.5 TEncoding.Convert

Synopsis: Convert an array of bytes from one encoding to another.

Declaration: `class function Convert(Source: TEncoding; Destination: TEncoding;
const Bytes: TBytes) : TBytes; Overload
class function Convert(Source: TEncoding; Destination: TEncoding;
const Bytes: TBytes; StartIndex: Integer;
Count: Integer) : TBytes; Overload`

Visibility: public

Description: Convert will convert the bytes in Bytes from encoding Source to encoding Destination and returns the converted bytes as an array. If StartIndex and Count are specified, the conversion starts at (zero-based) index StartIndex, and only Count bytes will be converted.

See also: TEncoding.GetBytes (1871), TEncoding.GetChars (1871)

76.96.6 TEncoding.IsStandardEncoding

Synopsis: Check if the encoding is one of the standard encodings.

Declaration: `class function IsStandardEncoding(AEncoding: TEncoding) : Boolean
; Static`

Visibility: public

Description: IsStandardEncoding will return True is one of the standard encoding instances TEncoding.ANSI (1876), TEncoding.ASCII (1876), TEncoding.Default (1876), TEncoding.BigEndianUnicode (1876), TEncoding.Unicode (1877), TEncoding.UTF7 (1877) or TEncoding.UTF8 (1878). Otherwise it returns False.

See also: TEncoding.ANSI (1876), TEncoding.ASCII (1876), TEncoding.Default (1876), TEncoding.BigEndianUnicode (1876), TEncoding.Unicode (1877), TEncoding.UTF7 (1877), TEncoding.UTF8 (1878)

76.96.7 TEncoding.GetBufferEncoding

Synopsis: Attempt to guess the encoding of a buffer.

Declaration: `class function GetBufferEncoding(const Buffer: TBytes;
var AEncoding: TEncoding) : Integer
; Overload; Static
class function GetBufferEncoding(const Buffer: TBytes;
var AEncoding: TEncoding;
ADefaultEncoding: TEncoding) : Integer
; Overload; Static`

Visibility: public

Description: GetBufferEncoding checks the BOM marker of an array of bytes Buffer, and returns the found encoding in AEncoding. Only the TEncoding.BigEndianUnicode (1876) TEncoding.Unicode (1877) and TEncoding.UTF8 (1878) encodings will be tried.

If AEncoding is non-Nil on entry, it will be used to check the BOM marker with. No other encoding will be tried in that case.

If ADefaultEncoding is specified, it will be returned in case no match was found.

The function returns the number of bytes in the BOM Marker.

See also: TEncoding.BigEndianUnicode (1876), TEncoding.Unicode (1877), TEncoding.UTF8 (1878), TEncoding.GetPreamble (1874)

76.96.8 TEncoding.GetEncoding

Synopsis: Get an encoding instance for a given codepage.

Declaration: `class function GetEncoding(CodePage: Integer) : TEncoding; Overload
; Static
class function GetEncoding(const EncodingName: UnicodeString)
: TEncoding; Overload; Static`

Visibility: public

Description: `GetEncoding` will return an instance of `TEncoding` for the given codepage `CodePage`. The codepage can also be specified by name `EncodingName`. The returned instance is an appropriate descendent of `TEncoding` and needs to be freed by the caller.

See also: `TUnicodeEncoding` ([1961](#)), `TBigEndianUnicodeEncoding` ([1846](#)), `TUTF8Encoding` ([1983](#)), `TUTF7Encoding` ([1982](#)), `TMBCSEncoding` ([1904](#))

76.96.9 TEncoding.GetMaxByteCount

Synopsis: Returns the maximum number of bytes needed to represent a string.

Declaration: `function GetMaxByteCount(CharCount: Integer) : Integer; Virtual
; Abstract`

Visibility: public

Description: `GetMaxByteCount` returns the maximum number of bytes needed to represent a string of `CharCount` characters in the given encoding.

This is an abstract method, implemented by descendents of `TEncoding`.

See also: `TEncoding.GetMaxCharCount` ([1874](#))

76.96.10 TEncoding.GetMaxCharCount

Synopsis: Return the maximum number of characters that can be represented in a number of bytes.

Declaration: `function GetMaxCharCount(ByteCount: Integer) : Integer; Virtual
; Abstract`

Visibility: public

Description: `GetMaxCharCount` returns the theoretical maximum number of characters that can be represented in a buffer of length `ByteCount` in the given encoding. Note that the actual number of characters that can be represented may well be much less and depends on the encoding and the actual characters.

This is an abstract method, implemented by descendents of `TEncoding`.

See also: `TEncoding.GetMaxByteCount` ([1874](#))

76.96.11 TEncoding.GetPreamble

Synopsis: Return the BOM Marker used by the encoding.

Declaration: `function GetPreamble : TBytes; Virtual; Abstract`

Visibility: public

Description: `GetPreamble` returns the BOM marker bytes used by the encoding. This is only meaningful for the UTF8 and Unicode encodings, for all other encodings there is no BOM Marker.

See also: `TUnicodeEncoding` ([1961](#)), `TBigEndianUnicodeEncoding` ([1846](#)), `TUTF8Encoding` ([1983](#))

76.96.12 TEncoding.GetString

Synopsis: Return a string based on an array of bytes.

Declaration: `function GetString(const Bytes: TBytes) : UnicodeString; Overload`
`function GetString(const Bytes: TBytes; ByteIndex: Integer;`
`ByteCount: Integer) : UnicodeString; Overload`

Visibility: public

Description: `GetString` will return a Unicode string, created from the bytes in the `Bytes` array. The bytes array will be interpreted according to the encoding which the `TEncoding` represents.

If `ByteIndex` and `ByteCount` are specified, only the `ByteCount` bytes starting at position `ByteIndex` will be converted.

Errors: In case of invalid bytes, an `EEncodingError` exception may be raised.

76.96.13 TEncoding.CodePage

Synopsis: Codepage for this encoding.

Declaration: `Property CodePage : Cardinal`

Visibility: public

Access: Read

Description: `CodePage` is the numerical codepage for this encoding. It is a number as used in the Windows codepage registry.

See also: `TEncoding.EncodingName` ([1875](#))

76.96.14 TEncoding.EncodingName

Synopsis: Name of this encoding.

Declaration: `Property EncodingName : UnicodeString`

Visibility: public

Access: Read

Description: `EncodingName` is the name for this encoding. It's based on the windows name for the encoding and is calculated from the codepage.

See also: `TEncoding.CodePage` ([1875](#))

76.96.15 TEncoding.IsSingleByte

Synopsis: Is the encoding a single-byte encoding or not ?

Declaration: `Property IsSingleByte : Boolean`

Visibility: public

Access: Read

Description: `IsSingleByte` determines whether an encoding is single-byte or not. It is `false` for all standard encodings.

76.96.16 TEncoding.ANSI

Synopsis: Ansi encoding instance.

Declaration: `Property ANSI : TEncoding`

Visibility: public

Access: Read

Description: `ANSI` is the ANSI codepage encoding instance, it is the default single-byte string codepage on windows (as returned by `DefaultSystemCodePage` (1609)). This instance is created and maintained by the system, it should not be freed.

See also: (??), `TEncoding.ASCII` (1876), `TEncoding.Default` (1876), `TEncoding.BigEndianUnicode` (1876), `TEncoding.Unicode` (1877), `TEncoding.UTF7` (1877), `TEncoding.UTF8` (1878)

76.96.17 TEncoding.ASCII

Synopsis: ASCII encoding instance.

Declaration: `Property ASCII : TEncoding`

Visibility: public

Access: Read

Description: `ASCII` is the ASCII codepage (`CP_ASCII`) encoding instance. This instance is created and maintained by the system, it should not be freed.

See also: `TEncoding.ANSI` (1876), `TEncoding.ASCII` (1876), `TEncoding.Default` (1876), `TEncoding.BigEndianUnicode` (1876), `TEncoding.Unicode` (1877), `TEncoding.UTF7` (1877), `TEncoding.UTF8` (1878)

76.96.18 TEncoding.BigEndianUnicode

Synopsis: Big-endian Unicode (UTF16BE) encoding instance.

Declaration: `Property BigEndianUnicode : TEncoding`

Visibility: public

Access: Read

Description: `BigEndianUnicode` is the Big-endian Unicode encoding instance (`CP_UTF16BE`) This instance is created and maintained by the system, it should not be freed.

See also: (??), `TEncoding.ANSI` (1876), `TEncoding.ASCII` (1876), `TEncoding.Default` (1876), `TEncoding.Unicode` (1877), `TEncoding.UTF7` (1877), `TEncoding.UTF8` (1878)

76.96.19 TEncoding.Default

Synopsis: Default codepage.

Declaration: `Property Default : TEncoding`

Visibility: public

Access: Read

Description: `Default` is the default encoding instance (it equals the ANSI codepage). This instance is created and maintained by the system, it should not be freed.

if the `DefaultSystemCodePage` (1415) changes, you should call `TEncoding.FreeEncodings` (1871) to regenerate the default encoding using the new code page.

See also: (??), `TEncoding.ANSI` (1876), `TEncoding.ASCII` (1876), `TEncoding.BigEndianUnicode` (1876), `TEncoding.Unicode` (1877), `TEncoding.UTF7` (1877), `TEncoding.UTF8` (1878)

76.96.20 `TEncoding.SystemEncoding`

Synopsis: System encoding.

Declaration: `Property SystemEncoding : TEncoding`

Visibility: public

Access: Read

Description: `SystemEncoding` is the encoding as referenced by the `DefaultSystemCodePage`.

See also: `DefaultSystemCodePage` (1415)

76.96.21 `TEncoding.Unicode`

Synopsis: UTF16 encoding instance.

Declaration: `Property Unicode : TEncoding`

Visibility: public

Access: Read

Description: `Unicode` is the Big-endian Unicode encoding instance (CP_UTF16) This instance is created and maintained by the system, it should not be freed.

See also: (??), `TEncoding.ANSI` (1876), `TEncoding.ASCII` (1876), `TEncoding.Default` (1876), `TEncoding.BigEndianUnicode` (1876), `TEncoding.Unicode` (1877), `TEncoding.UTF7` (1877), `TEncoding.UTF8` (1878)

76.96.22 `TEncoding.UTF7`

Synopsis: UTF7 encoding instance.

Declaration: `Property UTF7 : TEncoding`

Visibility: public

Access: Read

Description: `UTF7` is the UTF7 encoding instance (CP_UTF7) This instance is created and maintained by the system, it should not be freed.

See also: (??), `TEncoding.ANSI` (1876), `TEncoding.ASCII` (1876), `TEncoding.Default` (1876), `TEncoding.BigEndianUnicode` (1876), `TEncoding.Unicode` (1877), `TEncoding.UTF8` (1878)

76.96.23 TEncoding.UTF8

Synopsis: UTF8 encoding instance.

Declaration: `Property UTF8 : TEncoding`

Visibility: public

Access: Read

Description: UTF8 is the UTF7 encoding instance (CP_UTF7) This instance is created and maintained by the system, it should not be freed.

See also: [\(??\)](#), [TEncoding.ANSI \(1876\)](#), [TEncoding.ASCII \(1876\)](#), [TEncoding.Default \(1876\)](#), [TEncoding.BigEndianUnicode \(1876\)](#), [TEncoding.Unicode \(1877\)](#), [TEncoding.UTF7 \(1877\)](#)

76.97 TExtendedHelper

76.97.1 Description

`TExtendedHelper` is the helper type for the Extended-sized floating point type. It contains some conversion methods, as well as access to the low-level structure of the floating-point representation of a `Extended`.

See also: [TDoubleHelper \(1865\)](#), [TExtendedHelper \(1878\)](#)

76.97.2 Method overview

Page	Method	Description
1881	<code>BuildUp</code>	Build a Extended-sized floating point from its composing parts.
1882	<code>Exponent</code>	Exponent of the floating-point value.
1882	<code>Fraction</code>	Fraction of the floating-point value.
1880	<code>IsInfinity</code>	Check whether a value is positive or negative infinity.
1880	<code>IsNan</code>	Check whether a value equals NaN.
1881	<code>IsNegativeInfinity</code>	Check whether a value is negative infinity.
1881	<code>IsPositiveInfinity</code>	Check whether a value is positive infinity.
1882	<code>Mantissa</code>	Mantissa of the floating-point.
1879	<code>Parse</code>	Convert a string to a floating point value.
1881	<code>Size</code>	Size (in bytes) of a Extended-sized floating point value.
1882	<code>SpecialType</code>	Return the type of the Extended-sized floating point value.
1879	<code>ToString</code>	Convert a Extended-sized floating point value to a string.
1880	<code>TryParse</code>	Try to convert a string to a Extended-sized floating point value.

76.97.3 Property overview

Page	Properties	Access	Description
1883	<code>Bytes</code>	rw	Indexed access to the individual bytes of the floating point value.
1883	<code>Exp</code>	rw	The bit pattern of the exponent as stored in memory.
1884	<code>Frac</code>	rw	Bitpattern that makes up the fractional part.
1883	<code>Sign</code>	rw	Sign of the floating point value.
1883	<code>Words</code>	rw	Indexed access to the words that make up the floating point value.

76.97.4 TExtendedHelper.ToString

Synopsis: Convert a Extended-sized floating point value to a string.

```

Declaration: class function ToString(const AValue: Extended) : string; Overload
              ; Static
class function ToString(const AValue: Extended;
              const AFormatSettings: TFormatSettings) : string
              ; Overload; Static
class function ToString(const AValue: Extended;
              const AFormat: TFloatFormat;
              const APrecision: Integer;
              const ADigits: Integer) : string; Overload
              ; Static
class function ToString(const AValue: Extended;
              const AFormat: TFloatFormat;
              const APrecision: Integer;
              const ADigits: Integer;
              const AFormatSettings: TFormatSettings) : string
              ; Overload; Static
function ToString(const AFormat: TFloatFormat;
              const APrecision: Integer; const ADigits: Integer)
              : string; Overload
function ToString(const AFormat: TFloatFormat;
              const APrecision: Integer; const ADigits: Integer;
              const AFormatSettings: TFormatSettings) : string
              ; Overload
function ToString(const AFormatSettings: TFormatSettings) : string
              ; Overload
function ToString : string; Overload

```

Visibility: public

Description: ToString will convert AValue (or Self in the plain method version) to a string. Optionally FormatSettings can be specified, to be able to specify the decimal separator character to use.

Additionally, a precision `APrecision` and number of digits `ADigits` can be specified, in conjunction with a `AFormat` parameter to specify the form in which the floating-point value must be represented. (see `TFloatFormat` (1632) for an explanation of the various values). In this case, `FloatToStrF` (1710) is used to format the value. In the absence of these parameters, `FloatToStr` (1709) is called.

See also: [FloatToStr \(1709\)](#), [FloatToStrF \(1710\)](#), [TFloatFormat \(1632\)](#)

76.97.5 TExtendedHelper.Parse

Synopsis: Convert a string to a floating point value.

```
Declaration: class function Parse(const AString: string) : Extended; Overload
                ; Static
                class function Parse(const AString: string;
                                const AFormatSettings: TFormatSettings) : Extended
                ; Overload; Static
```

Visibility: public

Description: `Parse` will try to convert `AString` to a Extended-sized floating point value. It will take into account internationalization settings. (it uses `FloatToStr`).

Errors: If the string `AString` is not a valid floating-point value, a `EConvertError` ([1812](#)) exception is raised.

See also: `FloatToStr` ([1709](#)), `TExtendedHelper.ToString` ([1879](#)), `TExtendedHelper.TryParse` ([1880](#))

76.97.6 TExtendedHelper.TryParse

Synopsis: Try to convert a string to a Extended-sized floating point value.

Declaration:

```
class function TryParse(const AString: string; out AValue: Extended)
                        : Boolean; Overload; Static
class function TryParse(const AString: string; out AValue: Extended;
                        const AFormatSettings: TFormatSettings) : Boolean
                        ; Overload; Static
```

Visibility: public

Description: `TryParse` attempts to convert the string `AString` to a Extended-sized floating point value and reports `True` if the conversion was successful. In that case the parsed value is returned in `AValue`.

If the conversion failed, `False` is returned.

See also: `TExtendedHelper.Parse` ([1879](#)), `TExtendedHelper.ToString` ([1879](#))

76.97.7 TExtendedHelper.IsNaN

Synopsis: Check whether a value equals NaN.

Declaration:

```
class function IsNaN(const AValue: Extended) : Boolean; Overload
                        ; Static
function IsNaN : Boolean; Overload
```

Visibility: public

Description: `IsNaN` checks whether a Extended-sized floating point value is NaN (Not a Number). If so, it returns `True`. When the class function version is used, the value can be specified using `AValue`. In the method version, the used value is (`Self`).

See also: `TExtendedHelper.IsInfinity` ([1880](#)), `TExtendedHelper.IsPositiveInfinity` ([1881](#)), `TExtendedHelper.IsNegativeInfinity` ([1881](#))

76.97.8 TExtendedHelper.IsInfinity

Synopsis: Check whether a value is positive or negative infinity.

Declaration:

```
class function IsInfinity(const AValue: Extended) : Boolean; Overload
                        ; Static
function IsInfinity : Boolean; Overload
```

Visibility: public

Description: `IsInfinity` checks whether a Extended-sized floating point value represents a positive or negative infinity. If so, it returns `True`. When the class function version is used, the value can be specified using `AValue`. In the method version, the used value is (`Self`).

See also: `TExtendedHelper.IsNaN` ([1880](#)), `TExtendedHelper.IsPositiveInfinity` ([1881](#)), `TExtendedHelper.IsNegativeInfinity` ([1881](#))

76.97.9 TExtendedHelper.IsNegativeInfinity

Synopsis: Check whether a value is negative infinity.

Declaration: `class function IsNegativeInfinity(const AValue: Extended) : Boolean
; Overload; Static
function IsNegativeInfinity : Boolean; Overload`

Visibility: public

Description: `IsNegativeInfinity` checks whether a Extended-sized floating point value represents a negative infinity. If so, it returns `True`. When the class function version is used, the value can be specified using `AValue`. In the method version, the used value is `(Self)`.

See also: `TExtendedHelper.IsNan` ([1880](#)), `TExtendedHelper.IsPositiveInfinity` ([1881](#)), `TExtendedHelper.IsInfinity` ([1880](#))

76.97.10 TExtendedHelper.IsPositiveInfinity

Synopsis: Check whether a value is positive infinity.

Declaration: `class function IsPositiveInfinity(const AValue: Extended) : Boolean
; Overload; Static
function IsPositiveInfinity : Boolean; Overload`

Visibility: public

Description: `IsPositiveInfinity` checks whether a Extended-sized floating point value represents a positive infinity. If so, it returns `True`. When the class function version is used, the value can be specified using `AValue`. In the method version, the used value is `(Self)`.

See also: `TExtendedHelper.IsNan` ([1880](#)), `TExtendedHelper.IsNegativeInfinity` ([1881](#)), `TExtendedHelper.IsInfinity` ([1880](#))

76.97.11 TExtendedHelper.Size

Synopsis: Size (in bytes) of a Extended-sized floating point value.

Declaration: `class function Size : Integer; Static`

Visibility: public

Description: `Size` is the size (in bytes) of a Extended-sized floating point value. It is equivalent to calling `SizeOf(Extended)`.

See also: `SizeOf` ([1546](#))

76.97.12 TExtendedHelper.BuildUp

Synopsis: Build a Extended-sized floating point from its composing parts.

Declaration: `procedure BuildUp(const ASignFlag: Boolean; const AMantissa: QWord;
const AExponent: Integer)`

Visibility: public

Description: `BuildUp` will compose a Extended-sized floating point value from the sign `ASignFlag`, mantissa `AMantissa` and exponent `AExponent`. It simply sets the `Sign` ([1883](#)), `Exp` ([1883](#)) and `Frac` ([1884](#)) properties in 1 call.

See also: `TExtendedHelper.Sign` ([1883](#)), `TExtendedHelper.Exp` ([1883](#)), `TExtendedHelper.Frac` ([1884](#))

76.97.13 TExtendedHelper.Exponent

Synopsis: Exponent of the floating-point value.

Declaration: `function Exponent : Integer`

Visibility: `public`

Description: `Exponent` is the value X in the representation of the floating-point value in $m \cdot 2^X$, i.e. the exponent.

See also: `TExtendedHelper.Sign` ([1883](#)), `TExtendedHelper.Exp` ([1883](#)), `TExtendedHelper.Frac` ([1884](#)), `TExtendedHelper.Fraction` ([1882](#)), `TExtendedHelper.Mantissa` ([1882](#))

76.97.14 TExtendedHelper.Fraction

Synopsis: Fraction of the floating-point value.

Declaration: `function Fraction : Extended`

Visibility: `public`

Description: `Fraction` is the decimal part of the floating-point value.

See also: `TExtendedHelper.Sign` ([1883](#)), `TExtendedHelper.Exp` ([1883](#)), `TExtendedHelper.Exponent` ([1882](#)), `TExtendedHelper.Frac` ([1884](#)), `TExtendedHelper.Mantissa` ([1882](#))

76.97.15 TExtendedHelper.Mantissa

Synopsis: Mantissa of the floating-point.

Declaration: `function Mantissa : QWord`

Visibility: `public`

Description: `Mantissa` is the value of the significand without the hidden bit. This means it the plain bit pattern as it is stored in memory.

See also: `TExtendedHelper.Sign` ([1883](#)), `TExtendedHelper.Exp` ([1883](#)), `TExtendedHelper.Exponent` ([1882](#)), `TExtendedHelper.Frac` ([1884](#)), `TExtendedHelper.Fraction` ([1882](#))

76.97.16 TExtendedHelper.SpecialType

Synopsis: Return the type of the Extended-sized floating point value.

Declaration: `function SpecialType : TFloatSpecial`

Visibility: `public`

Description: `SpecialType` checks whether the Extended-sized floating point value equals one of several special values, and returns an enumerated value describing which value this is. See `TFloatSpecial` ([1609](#)) for an explanation of the possible values.

See also: `TFloatSpecial` ([1609](#))

76.97.17 TExtendedHelper.Bytes

Synopsis: Indexed access to the individual bytes of the floating point value.

Declaration: `Property Bytes[AIndex: Cardinal]: Byte`

Visibility: public

Access: Read,Write

Description: `Bytes` can be used to get or set the various bytes that make up the Extended-sized floating point value. The index runs from 0 to `Size-1`.

See also: `TExtendedHelper.Words` ([1883](#)), `TExtendedHelper.Size` ([1881](#))

76.97.18 TExtendedHelper.Words

Synopsis: Indexed access to the words that make up the floating point value.

Declaration: `Property Words[AIndex: Cardinal]: Word`

Visibility: public

Access: Read,Write

Description: `Words` can be used to get or set the various bytes that make up the Extended-sized floating point value. The index runs from 0 to $(\text{Size}-1) \div 2$.

See also: `TExtendedHelper.Bytes` ([1883](#)), `TExtendedHelper.Size` ([1881](#))

76.97.19 TExtendedHelper.Sign

Synopsis: Sign of the floating point value.

Declaration: `Property Sign : Boolean`

Visibility: public

Access: Read,Write

Description: `Sign` returns `True` if the sign bit of the value is set (i.e. it is a negative value) or `False` if it is not set (i.e. it is a positive value).

See also: `TExtendedHelper.Bytes` ([1883](#)), `TExtendedHelper.Exp` ([1883](#)), `TExtendedHelper.Frac` ([1884](#)), `TExtendedHelper.Mantissa` ([1882](#)), `TExtendedHelper.Fraction` ([1882](#)), `TExtendedHelper.Exponent` ([1882](#))

76.97.20 TExtendedHelper.Exp

Synopsis: The bit pattern of the exponent as stored in memory.

Declaration: `Property Exp : QWord`

Visibility: public

Access: Read,Write

Description: `Exp` is the internal representation of the Exponent ([1882](#)).

See also: `TExtendedHelper.Bytes` ([1883](#)), `TExtendedHelper.Sign` ([1883](#)), `TExtendedHelper.Frac` ([1884](#)), `TExtendedHelper.Mantissa` ([1882](#)), `TExtendedHelper.Fraction` ([1882](#)), `TExtendedHelper.Exponent` ([1882](#))

76.97.21 TExtendedHelper.Frac

Synopsis: Bitpattern that makes up the fractional part.

Declaration: `Property Frac : QWord`

Visibility: `public`

Access: `Read, Write`

Description: `Frac` is the bit pattern representing the fractional part (significand) including the preceding 1 (the hidden bit).

See also: `TExtendedHelper.Bytes` ([1883](#)), `TExtendedHelper.Sign` ([1883](#)), `TExtendedHelper.Exp` ([1883](#)), `TExtendedHelper.Mantissa` ([1882](#)), `TExtendedHelper.Fraction` ([1882](#)), `TExtendedHelper.Exponent` ([1882](#))

76.98 TGuidHelper**76.98.1 Description**

`TGuidHelper` provides various methods for a `TGUID` ([1609](#)) type.

See also: `TGUID` ([1609](#))

76.98.2 Method overview

Page	Method	Description
1884	<code>Create</code>	Initialize a <code>TGUID</code> instance from data.
1885	<code>NewGuid</code>	Create a new GUID.
1885	<code>ToByteArray</code>	Convert to array of bytes.
1885	<code>ToString</code>	Convert to string.

76.98.3 TGuidHelper.Create

Synopsis: Initialize a `TGUID` instance from data.

```
Declaration: class function Create(const Data; DataEndian: TEndian) : TGuid
                ; Overload; Static
class function Create(const B: TBytes; DataEndian: TEndian) : TGuid
                ; Overload; Static
class function Create(const B: TBytes; AStartIndex: Cardinal;
                DataEndian: TEndian) : TGuid; Overload; Static
class function Create(const S: string) : TGuid; Overload; Static
class function Create(A: Integer; B: SmallInt; C: SmallInt;
                const D: TBytes) : TGuid; Overload; Static
class function Create(A: Integer; B: SmallInt; C: SmallInt; D: Byte;
                E: Byte; F: Byte; G: Byte; H: Byte; I: Byte;
                J: Byte; K: Byte) : TGuid; Overload; Static
class function Create(A: Cardinal; B: Word; C: Word; D: Byte; E: Byte;
                F: Byte; G: Byte; H: Byte; I: Byte; J: Byte;
                K: Byte) : TGuid; Overload; Static
```

Visibility: `default`

Description: `Create` will initialize a `TGUID` ([1609](#)) from byte data. The byte-data can come in different forms

Data can be a plain buffer. The GUID Data will be read from the buffer (`SizeOf(TGUID)` bytes), according to the `BigEndian` argument.

Data can be an array of bytes. `SizeOf(TGUID)` bytes will be read from the array, starting at index `StartIndex` (default 0). The GUID Data will be read according to the `BigEndian` argument.

S can be a string containing a string representation of the GUID, which is converted using `StringToGUID` (1759).

A,B,C,D (optionally **E,F,G,H,I,J,K**) where the arguments are simply the various components of the GUID.

See also: `TGUID` (1609), `StringToGUID` (1759)

76.98.4 TGuidHelper.NewGuid

Synopsis: Create a new GUID.

Declaration: `class function NewGuid : TGuid; Static`

Visibility: default

Description: `NewGUID` creates and returns a new (??) using `CreateGUID` (1672)

See also: `CreateGUID` (1672)

76.98.5 TGuidHelper.ToByteArray

Synopsis: Convert to array of bytes.

Declaration: `function ToByteArray(DataEndian: TEndian) : TBytes`

Visibility: default

Description: `ToByteArray` converts the `TGUID` to an array of bytes (of length `SizeOf(TGUID)`). The bytes will be ordered according to `DataEndian`.

See also: `TGUIDHelper.ToString` (1885), `TGUIDHelper.Create` (1884)

76.98.6 TGuidHelper.ToString

Synopsis: Convert to string.

Declaration: `function ToString(SkipBrackets: Boolean) : string`

Visibility: default

Description: `ToString` will convert the `TGUID` to a string representation, using `GUIDToString` (1734)

See also: `GUIDToString` (1734)

76.99 TInt64Helper

76.99.1 Description

`TInt64Helper` contains some auxiliary routines for a `Int64`-typed ordinal value. It consists mainly of conversion routines to and from other types.

See also: `TStringHelper` ([1609](#)), `TShortIntHelper` ([1928](#)), `TSmallIntHelper` ([1955](#)), `TWordHelper` ([2000](#)), `TCardinalHelper` ([1857](#)), `TIntegerHelper` ([1893](#)), `TByteHelper` ([1851](#)), `TQWordHelper` ([1921](#)), `TNativeIntHelper` ([1908](#)), `TNativeUIntHelper` ([1915](#))

76.99.2 Method overview

Page	Method	Description
1890	<code>Clear</code>	Set value to 0.
1889	<code>ClearBit</code>	Set bit to 0.
1890	<code>HighestSetBitPos</code>	Return the position of the leftmost bit set.
1890	<code>LowestSetBitPos</code>	Return the position of the rightmost bit set.
1886	<code>Parse</code>	Convert from a string.
1889	<code>SetBit</code>	Set bit to 1.
1891	<code>SetBitsCount</code>	Count number of bits set.
1887	<code>Size</code>	Size, in bytes, of the <code>Int64</code> value.
1890	<code>TestBit</code>	Check bit.
1888	<code>ToBinString</code>	Convert to a binary string representation.
1887	<code>ToBoolean</code>	Convert to a boolean value.
1888	<code>ToDouble</code>	Convert to a double-sized floating point value.
1888	<code>ToExtended</code>	Convert to an extended-sized floating point value.
1889	<code>ToggleBit</code>	Invert bit.
1888	<code>ToHexString</code>	Convert to a hexadecimal string representation.
1889	<code>ToSingle</code>	Convert to a single-sized floating point value.
1887	<code>ToString</code>	Convert the value to string.
1887	<code>TryParse</code>	Try to convert a string to a <code>Int64</code> , report success or failure.

76.99.3 Property overview

Page	Properties	Access	Description
1891	<code>Bits</code>	rw	Read or write a bit.
1892	<code>Bytes</code>	rw	Read or write a byte.
1892	<code>DWords</code>	rw	Read or write a dword.
1891	<code>Nibbles</code>	rw	Read or write a nibble.
1892	<code>Words</code>	rw	Read or write a word.

76.99.4 TInt64Helper.Parse

Synopsis: Convert from a string.

Declaration: `class function Parse(const AString: string) : Int64; Static`

Visibility: `public`

Description: `Parse` will attempt to convert the string `AString` to a `Int64` value. It uses the `StrToInt64` ([1774](#)) function to perform the conversion, so no localization is taken into account.

Errors: If the string does not contain a valid `Int64` value, an `EConvertError` ([1812](#)) exception is raised.

See also: `TInt64Helper.ToString` ([1887](#)), `TInt64Helper.TryParse` ([1887](#)), `StrToInt64` ([1774](#))

76.99.5 `TInt64Helper.Size`

Synopsis: Size, in bytes, of the `Int64` value.

Declaration: `class function Size : Integer; Static`

Visibility: `public`

Description: `Size` returns the size (in `Int64`s) of the `Int64` value. This is equivalent to `SizeOf(Int64)`.

Errors: None.

See also: `SizeOf` ([1546](#))

76.99.6 `TInt64Helper.ToString`

Synopsis: Convert the value to string.

Declaration: `class function ToString(const AValue: Int64) : string; Overload
; Static
function ToString : string; Overload`

Visibility: `public`

Description: `ToString` will, in the class function variant of this method, convert `AValue` to a string representation. In the regular method overloaded version of `ToString`, the `Int64` value itself is used. The `IntToStr` ([1738](#)) function is used to do the conversion.

See also: `TInt64Helper.Parse` ([1886](#)), `IntToStr` ([1738](#))

76.99.7 `TInt64Helper.TryParse`

Synopsis: Try to convert a string to a `Int64`, report success or failure.

Declaration: `class function TryParse(const AString: string; out AValue: Int64)
: Boolean; Static`

Visibility: `public`

Description: `TryParse` attempts to convert the string `AString` to a `Int64`, and reports the success of the attempt. If the attempt is successful, then `True` is returned, and the actual value of the `Int64` is returned in `AValue`.

It uses the `val` ([1609](#)) function to perform the conversion, so no localization is taken into account.

See also: `TInt64Helper.Parse` ([1886](#)), `Val` ([1570](#))

76.99.8 `TInt64Helper.ToBoolean`

Synopsis: Convert to a boolean value.

Declaration: `function ToBoolean : Boolean`

Visibility: `public`

Description: `ToBoolean` converts the `Int64` value to a boolean: it returns `True` if the value is nonzero, `False` if it is zero.

See also: `TInt64Helper.ToSingle` (1889), `TInt64Helper.ToDouble` (1888), `TInt64Helper.ToExtended` (1888), `TInt64Helper.ToString` (1887), `TInt64Helper.ToHexString` (1888)

76.99.9 TInt64Helper.ToDouble

Synopsis: Convert to a double-sized floating point value.

Declaration: `function ToDouble : Double`

Visibility: `public`

Description: `ToDouble` converts the `Int64` value to a double-sized floating point value.

See also: `TInt64Helper.ToBoolean` (1887), `TInt64Helper.ToExtended` (1888), `TInt64Helper.ToSingle` (1889), `TInt64Helper.ToString` (1887), `TInt64Helper.ToHexString` (1888)

76.99.10 TInt64Helper.ToExtended

Synopsis: Convert to an extended-sized floating point value.

Declaration: `function ToExtended : Extended`

Visibility: `public`

Description: `ToDouble` converts the `Int64` value to an extended-sized floating point value.

See also: `TInt64Helper.ToBoolean` (1887), `TInt64Helper.ToSingle` (1889), `TInt64Helper.ToDouble` (1888), `TInt64Helper.ToString` (1887), `TInt64Helper.ToHexString` (1888)

76.99.11 TInt64Helper.ToBinString

Synopsis: Convert to a binary string representation.

Declaration: `function ToBinString : string`

Visibility: `public`

Description: `ToBinString` converts the `Int64` value to a binary string representation, showing all 64 bits.

See also: `TInt64Helper.ToHexString` (1888), `TInt64Helper.ToString` (1887)

76.99.12 TInt64Helper.ToHexString

Synopsis: Convert to a hexadecimal string representation.

Declaration: `function ToHexString(const AMinDigits: Integer) : string; Overload`
`function ToHexString : string; Overload`

Visibility: `public`

Description: `ToHexString` converts the `Int64` value to a hexadecimal string representation. The `AMinDigits` argument specifies the minimal number of characters in the resulting string. The string will be left-padded with zeroes if the representation contains less than `AMinDigits` characters.

See also: `TInt64Helper.ToBoolean` (1887), `TInt64Helper.ToSingle` (1889), `TInt64Helper.ToDouble` (1888), `TInt64Helper.ToString` (1887), `TInt64Helper.ToExtended` (1888)

76.99.13 TInt64Helper.ToSingle

Synopsis: Convert to an single-sized floating point value.

Declaration: `function ToSingle : Single`

Visibility: public

Description: `ToSingle` converts the `Int64` value to a single-sized floating point value.

See also: `TInt64Helper.ToBoolean` ([1887](#)), `TInt64Helper.ToDouble` ([1888](#)), `TInt64Helper.ToExtended` ([1888](#)), `TInt64Helper.ToString` ([1887](#)), `TInt64Helper.ToHexString` ([1888](#))

76.99.14 TInt64Helper.SetBit

Synopsis: Set bit to 1.

Declaration: `function SetBit(const Index: TInt64BitIndex) : Int64`

Visibility: public

Description: `SetBit` puts value 1 to bit number determined by argument `Index`.

See also: `TInt64Helper.ClearBit` ([1889](#)), `TInt64Helper.ToggleBit` ([1889](#)), `TInt64Helper.TestBit` ([1890](#)), `TInt64Helper.HighestSetBitPos` ([1890](#)), `TInt64Helper.LowestSetBitPos` ([1890](#)), `TInt64Helper.SetBitsCount` ([1891](#)), `TInt64Helper.Bits` ([1891](#))

76.99.15 TInt64Helper.ClearBit

Synopsis: Set bit to 0.

Declaration: `function ClearBit(const Index: TInt64BitIndex) : Int64`

Visibility: public

Description: `ClearBit` puts value 0 to bit number determined by argument `Index`.

See also: `TInt64Helper.SetBit` ([1889](#)), `TInt64Helper.ToggleBit` ([1889](#)), `TInt64Helper.TestBit` ([1890](#)), `TInt64Helper.HighestSetBitPos` ([1890](#)), `TInt64Helper.LowestSetBitPos` ([1890](#)), `TInt64Helper.SetBitsCount` ([1891](#)), `TInt64Helper.Bits` ([1891](#))

76.99.16 TInt64Helper.ToggleBit

Synopsis: Invert bit.

Declaration: `function ToggleBit(const Index: TInt64BitIndex) : Int64`

Visibility: public

Description: `ToggleBit` reads bit value from bit number determined by argument `Index`, inverts it (if it was 0 it becomes 1, and vice versa), and stores it back.

See also: `TInt64Helper.SetBit` ([1889](#)), `TInt64Helper.ToggleBit` ([1889](#)), `TInt64Helper.TestBit` ([1890](#)), `TInt64Helper.HighestSetBitPos` ([1890](#)), `TInt64Helper.LowestSetBitPos` ([1890](#)), `TInt64Helper.SetBitsCount` ([1891](#)), `TInt64Helper.Bits` ([1891](#))

76.99.17 TInt64Helper.TestBit

Synopsis: Check bit.

Declaration: `function TestBit (const Index: TInt64BitIndex) : Boolean`

Visibility: public

Description: `TestBit` reads boolean value (true if bit is 1, and false if bit is 0) from bit number determined by argument `Index`.

See also: `TInt64Helper.SetBit` (1889), `TInt64Helper.ToggleBit` (1889), `TInt64Helper.TestBit` (1890), `TInt64Helper.HighestSetBitPos` (1890), `TInt64Helper.LowestSetBitPos` (1890), `TInt64Helper.SetBitsCount` (1891), `TInt64Helper.Bits` (1891)

76.99.18 TInt64Helper.Clear

Synopsis: Set value to 0.

Declaration: `procedure Clear`

Visibility: public

76.99.19 TInt64Helper.HighestSetBitPos

Synopsis: Return the position of the leftmost bit set.

Declaration: `function HighestSetBitPos : Int8`

Visibility: public

Description: `HighestSetBitPos` scans the `Int64`, starting at the leftmost position and working towards position 0, and returns the index of the first set bit. The position is measured from the 0-th, rightmost bit. When there are no bits set, the result is -1.

See also: `System.BsrByte` (1609), `TInt64Helper.SetBit` (1889), `TInt64Helper.ClearBit` (1889), `TInt64Helper.ToggleBit` (1889), `TInt64Helper.TestBit` (1890), `TInt64Helper.LowestSetBitPos` (1890), `TInt64Helper.SetBitsCount` (1891), `TInt64Helper.Bits` (1891)

76.99.20 TInt64Helper.LowestSetBitPos

Synopsis: Return the position of the rightmost bit set.

Declaration: `function LowestSetBitPos : Int8`

Visibility: public

Description: `LowestSetBitPos` scans the `Int64`, starting at position 0 (rightmost position) and returns the index of the first set bit. The position is measured from the 0-th, rightmost bit. When there are no bits set, the result is -1.

See also: `System.BsfByte` (1609), `TInt64Helper.SetBit` (1889), `TInt64Helper.ClearBit` (1889), `TInt64Helper.ToggleBit` (1889), `TInt64Helper.TestBit` (1890), `TInt64Helper.HighestSetBitPos` (1890), `TInt64Helper.SetBitsCount` (1891), `TInt64Helper.Bits` (1891)

76.99.21 TInt64Helper.SetBitsCount

Synopsis: Count number of bits set.

Declaration: `function SetBitsCount : Byte`

Visibility: public

Description: `SetBitsCount` returns the number of bits set in an `Int64`.

See also: [System.PopCnt \(1609\)](#), [TInt64Helper.SetBit \(1889\)](#), [TInt64Helper.ClearBit \(1889\)](#), [TInt64Helper.ToggleBit \(1889\)](#), [TInt64Helper.TestBit \(1890\)](#), [TInt64Helper.HighestSetBitPos \(1890\)](#), [TInt64Helper.LowestSetBitPos \(1890\)](#), [TInt64Helper.Bits \(1891\)](#)

76.99.22 TInt64Helper.Bits

Synopsis: Read or write a bit.

Declaration: Property Bits[aIndex: TInt64BitIndex]: Boolean

Visibility: public

Access: Read, Write

Description: `Bits` property reads or writes boolean value to a bit number determined by argument `aIndex`.

[illegible]

See also: [TInt64Helper.SetBit \(1889\)](#), [TInt64Helper.ClearBit \(1889\)](#), [TInt64Helper.ToggleBit \(1889\)](#), [TInt64Helper.TestBit \(1890\)](#), [TInt64Helper.HighestSetBitPos \(1890\)](#), [TInt64Helper.LowestSetBitPos \(1890\)](#), [TInt64Helper.SetBitsCount \(1891\)](#), [TInt64Helper.Nibbles \(1891\)](#), [TInt64Helper.Bytes \(1892\)](#), [TInt64Helper.Words \(1892\)](#), [TInt64Helper.DWords \(1892\)](#)

76.99.23 TInt64Helper.Nibbles

Synopsis: Read or write a nibble.

Declaration: Property Nibbles[aIndex: TInt64NibbleIndex]: nibble

Visibility: public

Access: Read, Write

Description: `Nibbles` property reads or writes nibble value to a nibble number determined by argument `aIndex`.

[illegible]

See also: [TInt64Helper.Bits \(1891\)](#), [TInt64Helper.Bytes \(1892\)](#), [TInt64Helper.Words \(1892\)](#), [TInt64Helper.DWords \(1892\)](#)

76.99.24 TInt64Helper.Bytes

Synopsis: Read or write a byte.

Declaration: Property Bytes[aIndex: TInt64ByteIndex]: Byte

Visibility: public

Access: Read, Write

Description: Bytes property reads or writes byte value to a byte number determined by argument aIndex.

[illegible]

See also: TInt64Helper.Bits ([1891](#)), TInt64Helper.Nibbles ([1891](#)), TInt64Helper.Words ([1892](#)), TInt64Helper.DWords ([1892](#))

76.99.25 TInt64Helper.Words

Synopsis: Read or write a word.

Declaration: Property Words[aIndex: TInt64WordIndex]: Word

Visibility: public

Access: Read, Write

Description: Words property reads or writes word value to a word number determined by argument aIndex.

```
MyInt64.Clear; // %0000000000000000000000000000000000000000000000000000000000000000 1  
MyInt64.Words[1] := 1; // %000000000000000000000000000000000000000000000000000000001000000000  
WriteLn(MyInt64.Words[1]); // 1
```

See also: TInt64Helper.Bits ([1891](#)), TInt64Helper.Nibbles ([1891](#)), TInt64Helper.Bytes ([1892](#)), TInt64Helper.DWords ([1892](#))

76.99.26 TInt64Helper.DWords

Synopsis: Read or write a dword.

Declaration: Property DWords[aIndex: TInt64DwordIndex]: DWord

Visibility: public

Access: Read, Write

Description: DWords property reads or writes dword value to a dword number determined by argument aIndex.

[illegible]

See also: TInt64Helper.Bits ([1891](#)), TInt64Helper.Nibbles ([1891](#)), TInt64Helper.Bytes ([1892](#)), TInt64Helper.Words ([1892](#))

76.100 TIntegerHelper

76.100.1 Description

TIntegerHelper contains some auxiliary routines for a Integer-typed ordinal value. It consists mainly of conversion routines to and from other types.

See also: TStringHelper ([1609](#)), TShortIntHelper ([1928](#)), TSmallIntHelper ([1955](#)), TWordHelper ([2000](#)), TCardinalHelper ([1857](#)), TByteHelper ([1851](#)), TInt64Helper ([1886](#)), TQWordHelper ([1921](#)), TNativeIntHelper ([1908](#)), TNativeUIntHelper ([1915](#))

76.100.2 Method overview

Page	Method	Description
1897	Clear	Set value to 0.
1896	ClearBit	Set bit to 0.
1897	HighestSetBitPos	Return the position of the leftmost bit set.
1897	LowestSetBitPos	Return the position of the rightmost bit set.
1894	Parse	Convert from a string.
1896	SetBit	Set bit to 1.
1898	SetBitsCount	Count number of bits set.
1893	Size	Size, in bytes, of the Integer value.
1897	TestBit	Check bit.
1895	ToBinString	Convert to a binary string representation.
1894	ToBoolean	Convert to a boolean value.
1895	ToDouble	Convert to a double-sized floating point value.
1895	ToExtended	Convert to an extended-sized floating point value.
1896	ToggleBit	Invert bit.
1895	ToHexString	Convert to a hexadecimal string representation.
1896	ToSingle	Convert to an single-sized floating point value.
1894	ToString	Convert the value to string.
1894	TryParse	Try to convert a string to a Integer, report success or failure.

76.100.3 Property overview

Page	Properties	Access	Description
1898	Bits	rw	Read or write a bit.
1899	Bytes	rw	Read or write a byte.
1898	Nibbles	rw	Read or write a nibble.

76.100.4 TIntegerHelper.Size

Synopsis: Size, in bytes, of the Integer value.

Declaration: `class function Size : Integer; Static`

Visibility: `public`

Description: Size returns the size (in Integers) of the Integer value. This is equivalent to `SizeOf(Integer)`.

Errors: None.

See also: `SizeOf` ([1546](#))

76.100.5 TIntegerHelper.ToString

Synopsis: Convert the value to string.

```
Declaration: class function ToString(const AValue: Integer) : string; Overload
                ; Static
                function ToString : string; Overload
```

Visibility: public

Description: ToString will, in the class function variant of this method, convert AValue to a string representation. In the regular method overloaded version of ToString, the Integer value itself is used. The IntToStr (1738) function is used to do the conversion.

See also: [TIntegerHelper.Parse \(1894\)](#), [IntToStr \(1738\)](#)

76.100.6 TIntegerHelper.Parse

Synopsis: Convert from a string.

```
Declaration: class function Parse(const AString: string) : Integer; Static
```

Visibility: public

Description: Parse will attempt to convert the string AString to a Integer value. It uses the StrToInt (1773) function to perform the conversion, so no localization is taken into account.

Errors: If the string does not contain a valid Integer value, an `EConvertError` (1812) exception is raised.

See also: [TIntegerHelper.ToString \(1894\)](#), [TIntegerHelper.TryParse \(1894\)](#), [StrToInt \(1773\)](#)

76.100.7 TIntegerHelper.TryParse

Synopsis: Try to convert a string to a Integer, report success or failure.

```
Declaration: class function TryParse(const AString: string; out AValue: Integer)
              : Boolean; Static
```

Visibility: public

Description: TryParse attempts to convert the string AString to a Integer, and reports the success of the attempt. If the attempt is successful, then True is returned, and the actual value of the Integer is returned in AValue.

It uses the `val (1609)` function to perform the conversion, so no localization is taken into account.

See also: [TIntegerHelper.Parse \(1894\)](#), [Val \(1570\)](#)

76.100.8 TIntegerHelper.ToBoolean

Synopsis: Convert to a boolean value.

```
Declaration: function ToBoolean : Boolean
```

Visibility: public

Description: ToBoolean converts the Integer value to a boolean: it returns True if the value is nonzero, False if it is zero.

See also: [TIntegerHelper.ToSingle \(1896\)](#), [TIntegerHelper.ToDouble \(1895\)](#), [TIntegerHelper.ToExtended \(1895\)](#), [TIntegerHelper.ToString \(1894\)](#), [TIntegerHelper.ToHexString \(1895\)](#)

76.100.9 TIntegerHelper.ToDouble

Synopsis: Convert to a double-sized floating point value.

Declaration: `function ToDouble : Double`

Visibility: public

Description: `ToDouble` converts the `Integer` value to a double-sized floating point value.

See also: `TIntegerHelper.ToBoolean` (1894), `TIntegerHelper.ToExtended` (1895), `TIntegerHelper.ToSingle` (1896), `TIntegerHelper.ToString` (1894), `TIntegerHelper.ToHexString` (1895)

76.100.10 TIntegerHelper.ToExtended

Synopsis: Convert to an extended-sized floating point value.

Declaration: `function ToExtended : Extended`

Visibility: public

Description: `ToDouble` converts the `Integer` value to an extended-sized floating point value.

See also: `TIntegerHelper.ToBoolean` (1894), `TIntegerHelper.ToSingle` (1896), `TIntegerHelper.ToDouble` (1895), `TIntegerHelper.ToString` (1894), `TIntegerHelper.ToHexString` (1895)

76.100.11 TIntegerHelper.ToBinString

Synopsis: Convert to a binary string representation.

Declaration: `function ToBinString : string`

Visibility: public

Description: `ToBinString` converts the `Integer` value to a binary string representation, showing all 32 bits.

See also: `TIntegerHelper.ToHexString` (1895), `TIntegerHelper.ToString` (1894)

76.100.12 TIntegerHelper.ToHexString

Synopsis: Convert to a hexadecimal string representation.

Declaration: `function ToHexString(const AMinDigits: Integer) : string; Overload`
`function ToHexString : string; Overload`

Visibility: public

Description: `ToHexString` converts the `Integer` value to a hexadecimal string representation. The `AMinDigits` argument specifies the minimal number of characters in the resulting string. The string will be left-padded with zeroes if the representation contains less than `AMinDigits` characters.

See also: `TIntegerHelper.ToBoolean` (1894), `TIntegerHelper.ToSingle` (1896), `TIntegerHelper.ToDouble` (1895), `TIntegerHelper.ToString` (1894), `TIntegerHelper.ToExtended` (1895)

76.100.13 TIntegerHelper.ToSingle

Synopsis: Convert to an single-sized floating point value.

Declaration: `function ToSingle : Single`

Visibility: public

Description: `ToSingle` converts the Integer value to a single-sized floating point value.

See also: `TIntegerHelper.ToBoolean` ([1894](#)), `TIntegerHelper.ToDouble` ([1895](#)), `TIntegerHelper.ToExtended` ([1895](#)), `TIntegerHelper.ToString` ([1894](#)), `TIntegerHelper.ToHexString` ([1895](#))

76.100.14 TIntegerHelper.SetBit

Synopsis: Set bit to 1.

Declaration: `function SetBit(const Index: TIntegerBitIndex) : Integer`

Visibility: public

Description: `SetBit` puts value 1 to bit number determined by argument `Index`.

See also: `TIntegerHelper.ClearBit` ([1896](#)), `TIntegerHelper.ToggleBit` ([1896](#)), `TIntegerHelper.TestBit` ([1897](#)), `TIntegerHelper.HighestSetBitPos` ([1897](#)), `TIntegerHelper.LowestSetBitPos` ([1897](#)), `TIntegerHelper.SetBitsCount` ([1898](#)), `TIntegerHelper.Bits` ([1898](#))

76.100.15 TIntegerHelper.ClearBit

Synopsis: Set bit to 0.

Declaration: `function ClearBit(const Index: TIntegerBitIndex) : Integer`

Visibility: public

Description: `ClearBit` puts value 0 to bit number determined by argument `Index`.

See also: `TIntegerHelper.SetBit` ([1896](#)), `TIntegerHelper.ToggleBit` ([1896](#)), `TIntegerHelper.TestBit` ([1897](#)), `TIntegerHelper.HighestSetBitPos` ([1897](#)), `TIntegerHelper.LowestSetBitPos` ([1897](#)), `TIntegerHelper.SetBitsCount` ([1898](#)), `TIntegerHelper.Bits` ([1898](#))

76.100.16 TIntegerHelper.ToggleBit

Synopsis: Invert bit.

Declaration: `function ToggleBit(const Index: TIntegerBitIndex) : Integer`

Visibility: public

Description: `ToggleBit` reads bit value from bit number determined by argument `Index`, inverts it (if it was 0 it becomes 1, and vice versa), and stores it back.

See also: `TIntegerHelper.SetBit` ([1896](#)), `TIntegerHelper.ToggleBit` ([1896](#)), `TIntegerHelper.TestBit` ([1897](#)), `TIntegerHelper.HighestSetBitPos` ([1897](#)), `TIntegerHelper.LowestSetBitPos` ([1897](#)), `TIntegerHelper.SetBitsCount` ([1898](#)), `TIntegerHelper.Bits` ([1898](#))

76.100.17 TIntegerHelper.TestBit

Synopsis: Check bit.

Declaration: `function TestBit (const Index: TIntegerBitIndex) : Boolean`

Visibility: public

Description: `TestBit` reads boolean value (true if bit is 1, and false if bit is 0) from bit number determined by argument `Index`.

See also: `TIntegerHelper.SetBit` (1896), `TIntegerHelper.ToggleBit` (1896), `TIntegerHelper.TestBit` (1897), `TIntegerHelper.HighestSetBitPos` (1897), `TIntegerHelper.LowestSetBitPos` (1897), `TIntegerHelper.SetBitsCount` (1898), `TIntegerHelper.Bits` (1898)

76.100.18 TIntegerHelper.Clear

Synopsis: Set value to 0.

Declaration: `procedure Clear`

Visibility: public

76.100.19 TIntegerHelper.HighestSetBitPos

Synopsis: Return the position of the leftmost bit set.

Declaration: `function HighestSetBitPos : Int8`

Visibility: public

Description: `HighestSetBitPos` scans the Integer, starting at the leftmost position and working towards position 0, and returns the index of the first set bit. The position is measured from the 0-th, rightmost bit. When there are no bits set, the result is -1.

See also: `System.BsrByte` (1609), `TIntegerHelper.SetBit` (1896), `TIntegerHelper.ClearBit` (1896), `TIntegerHelper.ToggleBit` (1896), `TIntegerHelper.TestBit` (1897), `TIntegerHelper.LowestSetBitPos` (1897), `TIntegerHelper.SetBitsCount` (1898), `TIntegerHelper.Bits` (1898)

76.100.20 TIntegerHelper.LowestSetBitPos

Synopsis: Return the position of the rightmost bit set.

Declaration: `function LowestSetBitPos : Int8`

Visibility: public

Description: `LowestSetBitPos` scans the Integer, starting at position 0 (rightmost position) and returns the index of the first set bit. The position is measured from the 0-th, rightmost bit. When there are no bits set, the result is -1.

See also: `System.BsfByte` (1609), `TIntegerHelper.SetBit` (1896), `TIntegerHelper.ClearBit` (1896), `TIntegerHelper.ToggleBit` (1896), `TIntegerHelper.TestBit` (1897), `TIntegerHelper.HighestSetBitPos` (1897), `TIntegerHelper.SetBitsCount` (1898), `TIntegerHelper.Bits` (1898)

76.100.21 TIntegerHelper.SetBitsCount

Synopsis: Count number of bits set.

Declaration: `function SetBitsCount : Byte`

Visibility: `public`

Description: `SetBitsCount` returns the number of bits set in an `Integer`.

See also: `System.PopCnt` (1609), `TIntegerHelper.SetBit` (1896), `TIntegerHelper.ClearBit` (1896), `TIntegerHelper.ToggleBit` (1896), `TIntegerHelper.TestBit` (1897), `TIntegerHelper.HighestSetBitPos` (1897), `TIntegerHelper.LowestSetBitPos` (1897), `TIntegerHelper.Bits` (1898)

76.100.22 TIntegerHelper.Bits

Synopsis: Read or write a bit.

Declaration: `Property Bits[aIndex: TIntegerBitIndex]: Boolean`

Visibility: `public`

Access: `Read,Write`

Description: `Bits` property reads or writes boolean value to a bit number determined by argument `aIndex`.

```
MyInteger.Clear; // %00000000000000000000000000000000 MyInteger equals 0
MyInteger.Bits[0] := true; // %00000000000000000000000000000001 MyInteger equals 1
MyInteger.Bits[2] := true; // %00000000000000000000000000000101 MyInteger equals 5
WriteLn(MyInteger.Bits[2]); // TRUE
```

See also: `TIntegerHelper.SetBit` (1896), `TIntegerHelper.ClearBit` (1896), `TIntegerHelper.ToggleBit` (1896), `TIntegerHelper.TestBit` (1897), `TIntegerHelper.HighestSetBitPos` (1897), `TIntegerHelper.LowestSetBitPos` (1897), `TIntegerHelper.SetBitsCount` (1898), `TIntegerHelper.Nibbles` (1898), `TIntegerHelper.Bytes` (1899), `TIntegerHelper.Words` (1893)

76.100.23 TIntegerHelper.Nibbles

Synopsis: Read or write a nibble.

Declaration: `Property Nibbles[aIndex: TIntegerNibbleIndex]: nibble`

Visibility: `public`

Access: `Read,Write`

Description: `Nibbles` property reads or writes nibble value to a nibble number determined by argument `aIndex`.

```
MyInteger.Clear; // %00000000000000000000000000000000 MyInteger equals 0
MyInteger.Nibbles[1] := 3; // %0000000000000000000000000110000 MyInteger equals 48
WriteLn(MyInteger.Nibbles[1]); // 3
```

See also: `TIntegerHelper.Bits` (1898), `TIntegerHelper.Bytes` (1899), `TIntegerHelper.Words` (1893)

76.100.24 TIntegerHelper.Bytes

Synopsis: Read or write a byte.

Declaration: `Property Bytes[aIndex: TIntegerByteIndex]: Byte`

Visibility: `public`

Access: Read,Write

Description: Bytes property reads or writes byte value to a byte number determined by argument aIndex.

```

MyInteger.Clear; // %00000000000000000000000000000000 MyInteger equals 0
MyInteger.Bytes[1] := 1; // %000000000000000000000000000000001000000000 MyInteger equals 256
WriteLn(MyInteger.Bytes[1]); // 1

```

See also: TIntegerHelper.Bits ([1898](#)), TIntegerHelper.Nibbles ([1898](#)), TIntegerHelper.Words ([1893](#))

76.101 TLongBoolHelper

76.101.1 Description

TLongBoolHelper is a helper type for the LongBool type. It contains mostly conversion routines to and from other types.

See also: TStringHelper ([1609](#)), TShortIntHelper ([1928](#)), TSmallIntHelper ([1955](#)), TWordHelper ([2000](#)), TCardinalHelper ([1857](#)), TIntegerHelper ([1893](#)), TInt64Helper ([1886](#)), TQWordHelper ([1921](#)), TNativeIntHelper ([1908](#)), TByteHelper ([1851](#)), TByteBoolHelper ([1849](#)), TWordBoolHelper ([1998](#)), TLongBoolHelper ([1899](#))

76.101.2 Method overview

Page	Method	Description
1899	Parse	Convert string value to LongBool value.
1900	Size	Return the size (in bytes) of the.
1900	ToInteger	Convert to an integer value.
1900	ToString	Convert a LongBool value to string.
1900	TryToParse	Try to convert a string to a LongBool value.

76.101.3 TLongBoolHelper.Parse

Synopsis: Convert string value to LongBool value.

Declaration: `class function Parse(const S: string) : Boolean; Static`

Visibility: `public`

Description: Parse attempts to convert the string S to a LongBool value. It uses the StrToBool ([1768](#)) function to perform the conversion.

Errors: If S does not contain a valid string representation, then an EConvertError ([1812](#)) exception is raised.

See also: TLongBoolHelper.TryToParse ([1900](#)), TLongBoolHelper.ToString ([1900](#)), TLongBoolHelper.ToInteger ([1900](#))

76.101.4 TLongBoolHelper.Size

Synopsis: Return the size (in bytes) of the.

Declaration: `class function Size : Integer; Static`

Visibility: public

Description: `Size` returns the size (in bytes) of the `LongBool` value. This is equivalent to `SizeOf (LongBool)`.

See also: `SizeOf` ([1546](#))

76.101.5 TLongBoolHelper.ToString

Synopsis: Convert a `LongBool` value to string.

Declaration: `class function ToString(const AValue: Boolean;
UseBoolStrs: TUseBoolStrs) : string; Overload
; Static
function ToString(UseBoolStrs: TUseBoolStrs) : string; Overload`

Visibility: public

Description: `ToString` will, in the class method version, convert the `AValue` `LongBool` to a string representation. In the function method version the `LongBool` value itself (`Self`) will be converted.

If the `UseBoolStrs` parameter equals `TUseBoolStrs.True`, then the string representation will use the `LongBool` strings `BoolStrs` ([1609](#)). The default value for `UseBoolStrs` is `TUseBoolStrs.False`.

The conversion is done using the `BoolToStr` ([1664](#)) function.

See also: `BoolStrs` ([1609](#)), `BoolToStr` ([1664](#))

76.101.6 TLongBoolHelper.TryToParse

Synopsis: Try to convert a string to a `LongBool` value.

Declaration: `class function TryToParse(const S: string; out AValue: Boolean)
: Boolean; Static`

Visibility: public

Description: `TryToParse` will attempt to convert the string `S` to a `LongBool` value. If the attempt is successful, `True` is returned, and the actual value is returned in `AValue`. If the attempt failed, `False` is returned.

See also: `TLongBoolHelper.Parse` ([1899](#)), `TLongBoolHelper.ToString` ([1900](#))

76.101.7 TLongBoolHelper.ToInteger

Synopsis: Convert to an integer value.

Declaration: `function ToInteger : Integer`

Visibility: public

Description: `ToInteger` will return the `LongBool` value, typecasted to `Integer`.

See also: `TLongBoolHelper.ToString` ([1900](#))

76.102 TLongIntHelper

76.102.1 Method overview

Page	Method	Description
1903	Clear	
1903	ClearBit	
1903	HighestSetBitPos	
1903	LowestSetBitPos	
1901	Parse	
1902	SetBit	
1903	SetBitsCount	
1901	Size	
1903	TestBit	
1902	ToBinString	
1902	ToBoolean	
1902	ToDouble	
1902	ToExtended	
1903	ToggleBit	
1902	ToHexString	
1902	ToSingle	
1901	ToString	
1902	TryParse	

76.102.2 Property overview

Page	Properties	Access	Description
1903	Bits	rw	
1904	Bytes	rw	
1904	Nibbles	rw	
1904	Words	rw	

76.102.3 TLongIntHelper.Size

Declaration: `class function Size : Integer; Static`

Visibility: `public`

76.102.4 TLongIntHelper.ToString

Declaration: `class function ToString(const AValue: LongInt) : string; Overload
; Static
function ToString : string; Overload`

Visibility: `public`

76.102.5 TLongIntHelper.Parse

Declaration: `class function Parse(const AString: string) : LongInt; Static`

Visibility: `public`

76.102.6 TLongIntHelper.TryParse

Declaration: `class function TryParse(const AString: string; out AValue: LongInt)
: Boolean; Static`

Visibility: public

76.102.7 TLongIntHelper.ToBoolean

Declaration: `function ToBoolean : Boolean`

Visibility: public

76.102.8 TLongIntHelper.ToDouble

Declaration: `function ToDouble : Double`

Visibility: public

76.102.9 TLongIntHelper.ToExtended

Declaration: `function ToExtended : Extended`

Visibility: public

76.102.10 TLongIntHelper.ToBinString

Declaration: `function ToBinString : string`

Visibility: public

76.102.11 TLongIntHelper.ToHexString

Declaration: `function ToHexString(const AMinDigits: Integer) : string; Overload
function ToHexString : string; Overload`

Visibility: public

76.102.12 TLongIntHelper.ToSingle

Declaration: `function ToSingle : Single`

Visibility: public

76.102.13 TLongIntHelper.SetBit

Declaration: `function SetBit(const Index: TLongIntBitIndex) : LongInt`

Visibility: public

76.102.14 TLongIntHelper.ClearBit

Declaration: function ClearBit(const Index: TLongIntBitIndex) : LongInt

Visibility: public

76.102.15 TLongIntHelper.ToggleBit

Declaration: function ToggleBit(const Index: TLongIntBitIndex) : LongInt

Visibility: public

76.102.16 TLongIntHelper.TestBit

Declaration: function TestBit(const Index: TLongIntBitIndex) : Boolean

Visibility: public

76.102.17 TLongIntHelper.Clear

Declaration: procedure Clear

Visibility: public

76.102.18 TLongIntHelper.HighestSetBitPos

Declaration: function HighestSetBitPos : Int8

Visibility: public

76.102.19 TLongIntHelper.LowestSetBitPos

Declaration: function LowestSetBitPos : Int8

Visibility: public

76.102.20 TLongIntHelper.SetBitsCount

Declaration: function SetBitsCount : Byte

Visibility: public

76.102.21 TLongIntHelper.Bits

Declaration: Property Bits[aIndex: TLongIntBitIndex]: Boolean

Visibility: public

Access: Read,Write

76.102.22 TLongIntHelper.Nibbles

Declaration: `Property Nibbles[aIndex: TLongIntNibbleIndex]: nibble`

Visibility: `public`

Access: `Read,Write`

76.102.23 TLongIntHelper.Bytes

Declaration: `Property Bytes[aIndex: TLongIntByteIndex]: Byte`

Visibility: `public`

Access: `Read,Write`

76.102.24 TLongIntHelper.Words

Declaration: `Property Words[aIndex: TLongIntWordIndex]: Word`

Visibility: `public`

Access: `Read,Write`

76.103 TMBCSEncoding**76.103.1 Description**

`TMBCSEncoding` is the encoding class used for most of the encodings, except the actual Unicode encodings. The codepage is specified during creation, and is by default the system codepage.

The name is somewhat misleading, since the MBCS is also used for single-byte encodings.

See also: `TUnicodeEncoding` ([1961](#)), `TUTF8Encoding` ([1983](#)), `TUTF7Encoding` ([1982](#)), `TBigendianUnicodeEncoding` ([1846](#))

76.103.2 Method overview

Page	Method	Description
1905	<code>Clone</code>	Clone a <code>TMBCSEncoding</code> instance.
1904	<code>Create</code>	Create a new instance of a multi-byte character set encoding.
1905	<code>GetMaxByteCount</code>	Return max number of bytes needed to represent a string.
1905	<code>GetMaxCharCount</code>	Return max number of characters that can be represented by an array of bytes.
1905	<code>GetPreamble</code>	Return BOM marker bytes.

76.103.3 TMBCSEncoding.Create

Synopsis: Create a new instance of a multi-byte character set encoding.

Declaration: `constructor Create; Virtual; Overload`
`constructor Create(ACodePage: Integer); Virtual; Overload`
`constructor Create(ACodePage: Integer; MBToWCharFlags: Integer;`
`WCharToMBFlags: Integer); Virtual; Overload`

Visibility: public

Description: `Create` instantiates a new instance of the multi-byte character set encoding. The `ACodePage` parameter is optional, and defaults to `DefaultSystemCodePage` (1609).

The `MBToWCharFlags` and `WCharToMBFlags` parameters are stored but are otherwise unused in the Free Pascal implementation of `TMBCSEncoding`

76.103.4 `TMBCSEncoding.Clone`

Synopsis: Clone a `TMBCSEncoding` instance.

Declaration: `function Clone : TEncoding; Override`

Visibility: public

Description: `Clone` overrides `TEncoding.Clone` (1872) to provide a clone of the `TMBCSEncoding` instance.

See also: `TEncoding.Clone` (1872)

76.103.5 `TMBCSEncoding.GetMaxByteCount`

Synopsis: Return max number of bytes needed to represent a string.

Declaration: `function GetMaxByteCount (CharCount: Integer) : Integer; Override`

Visibility: public

Description: `GetMaxByteCount` overrides `TEncoding.GetMaxByteCount` (1874) to return the maximum number of bytes needed to represent a string.

See also: `TEncoding.GetMaxByteCount` (1874)

76.103.6 `TMBCSEncoding.GetMaxCharCount`

Synopsis: Return max number of characters that can be represented by an array of bytes.

Declaration: `function GetMaxCharCount (ByteCount: Integer) : Integer; Override`

Visibility: public

Description: `GetMaxCharCount` overrides `TEncoding.GetMaxCharCount` (1874) to return the maximum number of bytes needed to represent a string.

See also: `TEncoding.GetMaxCharCount` (1874)

76.103.7 `TMBCSEncoding.GetPreamble`

Synopsis: Return BOM marker bytes.

Declaration: `function GetPreamble : TBytes; Override`

Visibility: public

Description: `GetPreamble` overrides `TEncoding.GetPreamble` (1874) to return the BOM Marker bytes (none, for this implementation).

See also: `TEncoding.GetPreamble` (1874)

76.104 TMREWException

76.104.1 Description

TMREWException is the exception used by the TMultiReadExclusiveWriteSynchronizer (1906) class, for example in case of a deadlock.

See also: TMultiReadExclusiveWriteSynchronizer (1906)

76.105 TMultiReadExclusiveWriteSynchronizer

76.105.1 Description

TMultiReadExclusiveWriteSynchronizer is a default implementation of the IReadWriteSync (1825) interface. It uses a single mutex to protect access to the read/write resource, resulting in a single thread having access to the resource.

See also: IReadWriteSync (1825)

76.105.2 Interfaces overview

Page	Interfaces	Description
1825	IReadWriteSync	Read/Write synchronizer.

76.105.3 Method overview

Page	Method	Description
1907	Beginread	Request read access to the resource.
1907	Beginwrite	Request write access to the resource.
1906	Create	Create a new instance of the TMultiReadExclusiveWriteSynchronizer class.
1907	Destroy	Destroys the TMultiReadExclusiveWriteSynchronizer instance.
1908	Endread	Release read access to the resource.
1907	Endwrite	Release write access to the resource.

76.105.4 Property overview

Page	Properties	Access	Description
1908	RevisionLevel	r	
1908	WriterThreadID	r	

76.105.5 TMultiReadExclusiveWriteSynchronizer.Create

Synopsis: Create a new instance of the TMultiReadExclusiveWriteSynchronizer class.

Declaration: constructor Create; Virtual

Visibility: public

Description: Create creates a new instance of TMultiReadExclusiveWriteSynchronizer. It initializes a TRTLCriticalSection.

Errors: None.

See also: `TRTLCriticalSection` ([1396](#))

76.105.6 TMultiReadExclusiveWriteSynchronizer.Destroy

Synopsis: Destroys the `TMultiReadExclusiveWriteSynchronizer` instance.

Declaration: `destructor Destroy; Override`

Visibility: `public`

Description: Create destroys the instance of `TMultiReadExclusiveWriteSynchronizer`. It frees the `TRTLCriticalSection` it initialized, and calls the inherited destructor.

Errors: None.

See also: `TRTLCriticalSection` ([1396](#))

76.105.7 TMultiReadExclusiveWriteSynchronizer.Beginwrite

Synopsis: Request write access to the resource.

Declaration: `function Beginwrite : Boolean`

Visibility: `public`

Description: `Beginwrite` is the implementation of `IReadWriteSync.BeginWrite`. It simply enters the critical section, and returns `True`.

Errors: None.

See also: `IReadWriteSync.BeginWrite` ([1826](#)), `EndWrite` ([1907](#))

76.105.8 TMultiReadExclusiveWriteSynchronizer.Endwrite

Synopsis: Release write access to the resource.

Declaration: `procedure Endwrite`

Visibility: `public`

Description: `Beginwrite` is the implementation of `IReadWriteSync.EndWrite`. It simply leaves the critical section.

Errors: None.

See also: `IReadWriteSync.EndWrite` ([1826](#)), `BeginWrite` ([1907](#))

76.105.9 TMultiReadExclusiveWriteSynchronizer.Beginread

Synopsis: Request read access to the resource.

Declaration: `procedure Beginread`

Visibility: `public`

Description: `BeginRead` is the implementation of `IReadWriteSync.BeginRead`. It simply attempts to enter the critical section.

Errors: None.

See also: `IReadWriteSync.BeginRead` ([1825](#)), `EndRead` ([1908](#))

76.105.10 TMultiReadExclusiveWriteSynchronizer.Endread

Synopsis: Release read access to the resource.

Declaration: `procedure Endread`

Visibility: `public`

Description: `EndRead` is the implementation of `IReadWriteSync.EndRead`. It simply leaves the critical section.

Errors: None.

See also: `IReadWriteSync.EndRead` ([1825](#)), `BeginRead` ([1907](#))

76.105.11 TMultiReadExclusiveWriteSynchronizer.RevisionLevel

Declaration: `Property RevisionLevel : Cardinal`

Visibility: `public`

Access: `Read`

76.105.12 TMultiReadExclusiveWriteSynchronizer.WriterThreadID

Declaration: `Property WriterThreadID : TThreadID`

Visibility: `public`

Access: `Read`

76.106 TNativeIntHelper

76.106.1 Description

`TNativeIntHelper` contains some auxiliary routines for a `NativeInt`-typed ordinal value. It consists mainly of conversion routines to and from other types.

See also: `TStringHelper` ([1609](#)), `TShortIntHelper` ([1928](#)), `TSmallIntHelper` ([1955](#)), `TWordHelper` ([2000](#)), `TCardinalHelper` ([1857](#)), `TIntegerHelper` ([1893](#)), `TInt64Helper` ([1886](#)), `TQWordHelper` ([1921](#)), `TNativeIntHelper` ([1908](#)), `TByteHelper` ([1851](#))

76.106.2 Method overview

Page	Method	Description
1913	Clear	Set value to 0.
1912	ClearBit	Set bit to 0.
1913	HighestSetBitPos	Return the position of the leftmost bit set.
1913	LowestSetBitPos	Return the position of the rightmost bit set.
1909	Parse	Convert from a string.
1912	SetBit	Set bit to 1.
1913	SetBitsCount	Count number of bits set.
1909	Size	Size, in bytes, of the NativeInt value.
1912	TestBit	Check bit.
1911	ToBinString	Convert to a binary string representation.
1910	ToBoolean	Convert to a boolean value.
1910	ToDouble	Convert to a double-sized floating point value.
1911	ToExtended	Convert to an extended-sized floating point value.
1912	ToggleBit	Invert bit.
1911	ToHexString	Convert to a hexadecimal string representation.
1911	ToSingle	Convert to a single-sized floating point value.
1910	ToString	Convert the value to string.
1910	TryParse	Try to convert a string to a NativeInt, report success or failure.

76.106.3 Property overview

Page	Properties	Access	Description
1914	Bits	rw	Read or write a bit.
1914	Bytes	rw	Read or write a byte.
1914	Nibbles	rw	Read or write a nibble.

76.106.4 TNativeIntHelper.Parse

Synopsis: Convert from a string.

Declaration: `class function Parse(const AString: string) : NativeInt; Static`

Visibility: public

Description: `Parse` will attempt to convert the string `AString` to a `NativeInt` value. It uses the `StrToInt` ([1773](#)) function to perform the conversion, so no localization is taken into account.

Errors: If the string does not contain a valid `NativeInt` value, an `EConvertError` ([1812](#)) exception is raised.

See also: `TNativeIntHelper.ToString` ([1910](#)), `TNativeIntHelper.TryParse` ([1910](#)), `StrToInt` ([1773](#))

76.106.5 TNativeIntHelper.Size

Synopsis: Size, in bytes, of the `NativeInt` value.

Declaration: `class function Size : Integer; Static`

Visibility: public

Description: `Size` returns the size (in `NativeInts`) of the `NativeInt` value. This is equivalent to `SizeOf(NativeInt)`.

Errors: None.

See also: `SizeOf` ([1546](#))

76.106.6 TNativeIntHelper.ToString

Synopsis: Convert the value to string.

Declaration: `class function ToString(const AValue: NativeInt) : string; Overload
; Static
function ToString : string; Overload`

Visibility: public

Description: `ToString` will, in the class function variant of this method, convert `AValue` to a string representation. In the regular method overloaded version of `ToString`, the `NativeInt` value itself is used. The `IntToStr` (1738) function is used to do the conversion.

See also: `TNativeIntHelper.Parse` (1909), `IntToStr` (1738)

76.106.7 TNativeIntHelper.TryParse

Synopsis: Try to convert a string to a `NativeInt`, report success or failure.

Declaration: `class function TryParse(const AString: string; out AValue: NativeInt)
: Boolean; Static`

Visibility: public

Description: `TryParse` attempts to convert the string `AString` to a `NativeInt`, and reports the success of the attempt. If the attempt is successful, then `True` is returned, and the actual value of the `NativeInt` is returned in `AValue`.

It uses the `val` (1609) function to perform the conversion, so no localization is taken into account.

See also: `TNativeIntHelper.Parse` (1909), `Val` (1570)

76.106.8 TNativeIntHelper.ToBoolean

Synopsis: Convert to a boolean value.

Declaration: `function ToBoolean : Boolean`

Visibility: public

Description: `ToBoolean` converts the `NativeInt` value to a boolean: it returns `True` if the value is nonzero, `False` if it is zero.

See also: `TNativeIntHelper.ToSingle` (1911), `TNativeIntHelper.ToDouble` (1910), `TNativeIntHelper.ToExtended` (1911), `TNativeIntHelper.ToString` (1910), `TNativeIntHelper.ToHexString` (1911)

76.106.9 TNativeIntHelper.ToDouble

Synopsis: Convert to a double-sized floating point value.

Declaration: `function ToDouble : Double`

Visibility: public

Description: `ToDouble` converts the `NativeInt` value to a double-sized floating point value.

See also: `TNativeIntHelper.ToBoolean` (1910), `TNativeIntHelper.ToExtended` (1911), `TNativeIntHelper.ToSingle` (1911), `TNativeIntHelper.ToString` (1910), `TNativeIntHelper.ToHexString` (1911)

76.106.10 TNativeIntHelper.ToExtended

Synopsis: Convert to an extended-sized floating point value.

Declaration: `function ToExtended : Extended`

Visibility: public

Description: `ToDouble` converts the `NativeInt` value to an extended-sized floating point value.

See also: `TNativeIntHelper.ToBoolean` ([1910](#)), `TNativeIntHelper.ToSingle` ([1911](#)), `TNativeIntHelper.ToDouble` ([1910](#)), `TNativeIntHelper.ToString` ([1910](#)), `TNativeIntHelper.ToHexString` ([1911](#))

76.106.11 TNativeIntHelper.ToBinString

Synopsis: Convert to a binary string representation.

Declaration: `function ToBinString : string`

Visibility: public

Description: `ToBinString` converts the `NativeInt` value to a binary string representation, showing all bits.

See also: `TNativeIntHelper.ToHexString` ([1911](#)), `TNativeIntHelper.ToString` ([1910](#))

76.106.12 TNativeIntHelper.ToHexString

Synopsis: Convert to a hexadecimal string representation.

Declaration: `function ToHexString(const AMinDigits: Integer) : string; Overload`
`function ToHexString : string; Overload`

Visibility: public

Description: `ToHexString` converts the `NativeInt` value to a hexadecimal string representation. The `AMinDigits` argument specifies the minimal number of characters in the resulting string. The string will be left-padded with zeroes if the representation contains less than `AMinDigits` characters.

See also: `TNativeIntHelper.ToBoolean` ([1910](#)), `TNativeIntHelper.ToSingle` ([1911](#)), `TNativeIntHelper.ToDouble` ([1910](#)), `TNativeIntHelper.ToString` ([1910](#)), `TNativeIntHelper.ToExtended` ([1911](#))

76.106.13 TNativeIntHelper.ToSingle

Synopsis: Convert to an single-sized floating point value.

Declaration: `function ToSingle : Single`

Visibility: public

Description: `ToSingle` converts the `NativeInt` value to a single-sized floating point value.

See also: `TNativeIntHelper.ToBoolean` ([1910](#)), `TNativeIntHelper.ToDouble` ([1910](#)), `TNativeIntHelper.ToExtended` ([1911](#)), `TNativeIntHelper.ToString` ([1910](#)), `TNativeIntHelper.ToHexString` ([1911](#))

76.106.14 TNativeIntHelper.SetBit

Synopsis: Set bit to 1.

Declaration: `function SetBit(const Index: TNativeIntBitIndex) : NativeInt`

Visibility: public

Description: `SetBit` puts value 1 to bit number determined by argument `Index`.

See also: `TNativeIntHelper.ClearBit` (1912), `TNativeIntHelper.ToggleBit` (1912), `TNativeIntHelper.TestBit` (1912), `TNativeIntHelper.HighestSetBitPos` (1913), `TNativeIntHelper.LowestSetBitPos` (1913), `TNativeIntHelper.SetBitsCount` (1913), `TNativeIntHelper.Bits` (1914)

76.106.15 TNativeIntHelper.ClearBit

Synopsis: Set bit to 0.

Declaration: `function ClearBit(const Index: TNativeIntBitIndex) : NativeInt`

Visibility: public

Description: `ClearBit` puts value 0 to bit number determined by argument `Index`.

See also: `TNativeIntHelper.SetBit` (1912), `TNativeIntHelper.ToggleBit` (1912), `TNativeIntHelper.TestBit` (1912), `TNativeIntHelper.HighestSetBitPos` (1913), `TNativeIntHelper.LowestSetBitPos` (1913), `TNativeIntHelper.SetBitsCount` (1913), `TNativeIntHelper.Bits` (1914)

76.106.16 TNativeIntHelper.ToggleBit

Synopsis: Invert bit.

Declaration: `function ToggleBit(const Index: TNativeIntBitIndex) : NativeInt`

Visibility: public

Description: `ToggleBit` reads bit value from bit number determined by argument `Index`, inverts it (if it was 0 it becomes 1, and vice versa), and stores it back.

See also: `TNativeIntHelper.SetBit` (1912), `TNativeIntHelper.ToggleBit` (1912), `TNativeIntHelper.TestBit` (1912), `TNativeIntHelper.HighestSetBitPos` (1913), `TNativeIntHelper.LowestSetBitPos` (1913), `TNativeIntHelper.SetBitsCount` (1913), `TNativeIntHelper.Bits` (1914)

76.106.17 TNativeIntHelper.TestBit

Synopsis: Check bit.

Declaration: `function TestBit(const Index: TNativeIntBitIndex) : Boolean`

Visibility: public

Description: `TestBit` reads boolean value (true if bit is 1, and false if bit is 0) from bit number determined by argument `Index`.

See also: `TNativeIntHelper.SetBit` (1912), `TNativeIntHelper.ToggleBit` (1912), `TNativeIntHelper.TestBit` (1912), `TNativeIntHelper.HighestSetBitPos` (1913), `TNativeIntHelper.LowestSetBitPos` (1913), `TNativeIntHelper.SetBitsCount` (1913), `TNativeIntHelper.Bits` (1914)

76.106.18 TNativeIntHelper.Clear

Synopsis: Set value to 0.

Declaration: `procedure Clear`

Visibility: `public`

76.106.19 TNativeIntHelper.HighestSetBitPos

Synopsis: Return the position of the leftmost bit set.

Declaration: `function HighestSetBitPos : Int8`

Visibility: `public`

Description: `HighestSetBitPos` scans the `NativeInt`, starting at the leftmost position and working towards position 0, and returns the index of the first set bit. The position is measured from the 0-th, rightmost bit. When there are no bits set, the result is -1.

See also: `System.BsrByte` ([1609](#)), `TNativeIntHelper.SetBit` ([1912](#)), `TNativeIntHelper.ClearBit` ([1912](#)), `TNativeIntHelper.ToggleBit` ([1912](#)), `TNativeIntHelper.TestBit` ([1912](#)), `TNativeIntHelper.LowestSetBitPos` ([1913](#)), `TNativeIntHelper.SetBitsCount` ([1913](#)), `TNativeIntHelper.Bits` ([1914](#))

76.106.20 TNativeIntHelper.LowestSetBitPos

Synopsis: Return the position of the rightmost bit set.

Declaration: `function LowestSetBitPos : Int8`

Visibility: `public`

Description: `LowestSetBitPos` scans the `NativeInt`, starting at position 0 (rightmost position) and returns the index of the first set bit. The position is measured from the 0-th, rightmost bit. When there are no bits set, the result is -1.

See also: `System.BsfByte` ([1609](#)), `TNativeIntHelper.SetBit` ([1912](#)), `TNativeIntHelper.ClearBit` ([1912](#)), `TNativeIntHelper.ToggleBit` ([1912](#)), `TNativeIntHelper.TestBit` ([1912](#)), `TNativeIntHelper.HighestSetBitPos` ([1913](#)), `TNativeIntHelper.SetBitsCount` ([1913](#)), `TNativeIntHelper.Bits` ([1914](#))

76.106.21 TNativeIntHelper.SetBitsCount

Synopsis: Count number of bits set.

Declaration: `function SetBitsCount : Byte`

Visibility: `public`

Description: `SetBitsCount` returns the number of bits set in a `NativeInt`.

See also: `System.PopCnt` ([1609](#)), `TNativeIntHelper.SetBit` ([1912](#)), `TNativeIntHelper.ClearBit` ([1912](#)), `TNativeIntHelper.ToggleBit` ([1912](#)), `TNativeIntHelper.TestBit` ([1912](#)), `TNativeIntHelper.HighestSetBitPos` ([1913](#)), `TNativeIntHelper.LowestSetBitPos` ([1913](#)), `TNativeIntHelper.Bits` ([1914](#))

76.106.22 TNativeIntHelper.Bits

Synopsis: Read or write a bit.

Declaration: Property Bits[aIndex: TNativeIntBitIndex]: Boolean

Visibility: public

Access: Read,Write

Description: Bits property reads or writes boolean value to a bit number determined by argument aIndex.

```
MyNativeInt.Clear; // MyNativeInt equals 0
MyNativeInt.Bits[0] := true; // MyNativeInt equals 1
MyNativeInt.Bits[2] := true; // MyNativeInt equals 5
WriteLn(MyNativeInt.Bits[2]); // TRUE
```

See also: TNativeIntHelper.SetBit ([1912](#)), TNativeIntHelper.ClearBit ([1912](#)), TNativeIntHelper.ToggleBit ([1912](#)), TNativeIntHelper.TestBit ([1912](#)), TNativeIntHelper.HighestSetBitPos ([1913](#)), TNativeIntHelper.LowestSetBitPos ([1913](#)), TNativeIntHelper.SetBitsCount ([1913](#)), TNativeIntHelper.Nibbles ([1914](#)), TNativeIntHelper.Bytes ([1914](#)), TNativeIntHelper.Words ([1908](#)), TNativeIntHelper.DWords ([1908](#))

76.106.23 TNativeIntHelper.Nibbles

Synopsis: Read or write a nibble.

Declaration: Property Nibbles[aIndex: TNativeIntNibbleIndex]: nibble

Visibility: public

Access: Read,Write

Description: Nibbles property reads or writes nibble value to a nibble number determined by argument aIndex.

```
MyNativeInt.Clear; // MyNativeInt equals 0
MyNativeInt.Nibbles[1] := 3; // MyNativeInt equals 48
WriteLn(MyNativeInt.Nibbles[1]); // 3
```

See also: TNativeIntHelper.Bits ([1914](#)), TNativeIntHelper.Bytes ([1914](#)), TNativeIntHelper.Words ([1908](#)), TNativeIntHelper.DWords ([1908](#))

76.106.24 TNativeIntHelper.Bytes

Synopsis: Read or write a byte.

Declaration: Property Bytes[aIndex: TNativeIntByteIndex]: Byte

Visibility: public

Access: Read,Write

Description: Bytes property reads or writes byte value to a byte number determined by argument aIndex.

```
MyNativeInt.Clear; // MyNativeInt equals 0
MyNativeInt.Bytes[1] := 1; // MyNativeInt equals 256
WriteLn(MyNativeInt.Bytes[1]); // 1
```

See also: TNativeIntHelper.Bits ([1914](#)), TNativeIntHelper.Nibbles ([1914](#)), TNativeIntHelper.Words ([1908](#)), TNativeIntHelper.DWords ([1908](#))

76.107 TNativeUIntHelper

76.107.1 Description

`NativeUInt` contains some auxiliary routines for a `NativeUInt`-typed ordinal value. It consists mainly of conversion routines to and from other types.

See also: `TStringHelper` (1609), `TShortIntHelper` (1928), `TSmallIntHelper` (1955), `TWordHelper` (2000), `TCardinalHelper` (1857), `TIntegerHelper` (1893), `TInt64Helper` (1886), `TQWordHelper` (1921), `TNativeIntHelper` (1908), `TByteHelper` (1851)

76.107.2 Method overview

Page	Method	Description
1919	<code>Clear</code>	Set value to 0.
1918	<code>ClearBit</code>	Set bit to 0.
1919	<code>HighestSetBitPos</code>	Return the position of the leftmost bit set.
1919	<code>LowestSetBitPos</code>	Return the position of the rightmost bit set.
1915	<code>Parse</code>	Convert from a string.
1918	<code>SetBit</code>	Set bit to 1.
1920	<code>SetBitsCount</code>	Count number of bits set.
1916	<code>Size</code>	Size, in bytes, of the <code>NativeUInt</code> value.
1919	<code>TestBit</code>	Check bit.
1917	<code>ToBinString</code>	Convert to a binary string representation.
1916	<code>ToBoolean</code>	Convert to a boolean value.
1917	<code>ToDouble</code>	Convert to a double-sized floating point value.
1917	<code>ToExtended</code>	Convert to an extended-sized floating point value.
1918	<code>ToggleBit</code>	Invert bit.
1917	<code>ToHexString</code>	Convert to a hexadecimal string representation.
1918	<code>ToSingle</code>	Convert to a single-sized floating point value.
1916	<code>ToString</code>	Convert the value to string.
1916	<code>TryParse</code>	Try to convert a string to a <code>NativeUInt</code> , report success or failure.

76.107.3 Property overview

Page	Properties	Access	Description
1920	<code>Bits</code>	rw	Read or write a bit.
1921	<code>Bytes</code>	rw	Read or write a byte.
1920	<code>Nibbles</code>	rw	Read or write a nibble.

76.107.4 TNativeUIntHelper.Parse

Synopsis: Convert from a string.

Declaration: `class function Parse(const AString: string) : NativeUInt; Static`

Visibility: `public`

Description: `Parse` will attempt to convert the string `AString` to a `NativeUInt` value. It uses the `StrToInt` (1773) function to perform the conversion, so no localization is taken into account.

Errors: If the string does not contain a valid `NativeUInt` value, an `EConvertError` (1812) exception is raised.

See also: `TNativeUIntHelper.ToString` (1916), `TNativeUIntHelper.TryParse` (1916), `StrToInt` (1773)

76.107.5 TNativeUIntHelper.Size

Synopsis: Size, in bytes, of the NativeUInt value.

Declaration: `class function Size : Integer; Static`

Visibility: public

Description: `Size` returns the size (in NativeUInts) of the NativeUInt value. This is equivalent to `SizeOf (NativeUInt)`.

Errors: None.

See also: `SizeOf` ([1546](#))

76.107.6 TNativeUIntHelper.ToString

Synopsis: Convert the value to string.

Declaration: `class function ToString(const AValue: NativeUInt) : string; Overload
; Static
function ToString : string; Overload`

Visibility: public

Description: `ToString` will, in the class function variant of this method, convert `AValue` to a string representation. In the regular method overloaded version of `ToString`, the NativeUInt value itself is used. The `IntToStr` ([1738](#)) function is used to do the conversion.

See also: `TNativeUIntHelper.Parse` ([1915](#)), `IntToStr` ([1738](#))

76.107.7 TNativeUIntHelper.TryParse

Synopsis: Try to convert a string to a NativeUInt, report success or failure.

Declaration: `class function TryParse(const AString: string; out AValue: NativeUInt)
: Boolean; Static`

Visibility: public

Description: `TryParse` attempts to convert the string `AString` to a NativeUInt, and reports the success of the attempt. If the attempt is successful, then `True` is returned, and the actual value of the NativeUInt is returned in `AValue`.

It uses the `val` ([1609](#)) function to perform the conversion, so no localization is taken into account.

See also: `TNativeUIntHelper.Parse` ([1915](#)), `Val` ([1570](#))

76.107.8 TNativeUIntHelper.ToBoolean

Synopsis: Convert to a boolean value.

Declaration: `function ToBoolean : Boolean`

Visibility: public

Description: `ToBoolean` converts the NativeUInt value to a boolean: it returns `True` if the value is nonzero, `False` if it is zero.

See also: `TNativeUIntHelper.ToSingle` ([1918](#)), `TNativeUIntHelper.ToDouble` ([1917](#)), `TNativeUIntHelper.ToExtended` ([1917](#)), `TNativeUIntHelper.ToString` ([1916](#)), `TNativeUIntHelper.ToHexString` ([1917](#))

76.107.9 TNativeUIntHelper.ToDouble

Synopsis: Convert to a double-sized floating point value.

Declaration: `function ToDouble : Double`

Visibility: public

Description: `ToDouble` converts the `NativeUInt` value to a double-sized floating point value.

See also: `TNativeUIntHelper.ToBoolean` (1916), `TNativeUIntHelper.ToExtended` (1917), `TNativeUIntHelper.ToSingle` (1918), `TNativeUIntHelper.ToString` (1916), `TNativeUIntHelper.ToHexString` (1917)

76.107.10 TNativeUIntHelper.ToExtended

Synopsis: Convert to an extended-sized floating point value.

Declaration: `function ToExtended : Extended`

Visibility: public

Description: `ToDouble` converts the `NativeUInt` value to an extended-sized floating point value.

See also: `TNativeUIntHelper.ToBoolean` (1916), `TNativeUIntHelper.ToSingle` (1918), `TNativeUIntHelper.ToDouble` (1917), `TNativeUIntHelper.ToString` (1916), `TNativeUIntHelper.ToHexString` (1917)

76.107.11 TNativeUIntHelper.ToBinString

Synopsis: Convert to a binary string representation.

Declaration: `function ToBinString : string`

Visibility: public

Description: `ToBinString` converts the `NativeUInt` value to a binary string representation, showing all bits.

76.107.12 TNativeUIntHelper.ToHexString

Synopsis: Convert to a hexadecimal string representation.

Declaration: `function ToHexString(const AMinDigits: Integer) : string; Overload`
`function ToHexString : string; Overload`

Visibility: public

Description: `ToHexString` converts the `NativeUInt` value to a hexadecimal string representation. The `AMinDigits` argument specifies the minimal number of characters in the resulting string. The string will be left-padded with zeroes if the representation contains less than `AMinDigits` characters.

See also: `TNativeUIntHelper.ToBoolean` (1916), `TNativeUIntHelper.ToSingle` (1918), `TNativeUIntHelper.ToDouble` (1917), `TNativeUIntHelper.ToString` (1916), `TNativeUIntHelper.ToExtended` (1917)

76.107.13 TNativeUIntHelper.ToSingle

Synopsis: Convert to an single-sized floating point value.

Declaration: `function ToSingle : Single`

Visibility: public

Description: `ToSingle` converts the `NativeUInt` value to a single-sized floating point value.

See also: `TNativeUIntHelper.ToBoolean` (1916), `TNativeUIntHelper.ToDouble` (1917), `TNativeUIntHelper.ToExtended` (1917), `TNativeUIntHelper.ToString` (1916), `TNativeUIntHelper.ToHexString` (1917)

76.107.14 TNativeUIntHelper.SetBit

Synopsis: Set bit to 1.

Declaration: `function SetBit(const Index: TNativeUIntBitIndex) : NativeUInt`

Visibility: public

Description: `SetBit` puts value 1 to bit number determined by argument `Index`.

See also: `TNativeUIntHelper.ClearBit` (1918), `TNativeUIntHelper.ToggleBit` (1918), `TNativeUIntHelper.TestBit` (1919), `TNativeUIntHelper.HighestSetBitPos` (1919), `TNativeUIntHelper.LowestSetBitPos` (1919), `TNativeUIntHelper.SetBitsCount` (1920), `TNativeUIntHelper.Bits` (1920)

76.107.15 TNativeUIntHelper.ClearBit

Synopsis: Set bit to 0.

Declaration: `function ClearBit(const Index: TNativeUIntBitIndex) : NativeUInt`

Visibility: public

Description: `ClearBit` puts value 0 to bit number determined by argument `Index`.

See also: `TNativeUIntHelper.SetBit` (1918), `TNativeUIntHelper.ToggleBit` (1918), `TNativeUIntHelper.TestBit` (1919), `TNativeUIntHelper.HighestSetBitPos` (1919), `TNativeUIntHelper.LowestSetBitPos` (1919), `TNativeUIntHelper.SetBitsCount` (1920), `TNativeUIntHelper.Bits` (1920)

76.107.16 TNativeUIntHelper.ToggleBit

Synopsis: Invert bit.

Declaration: `function ToggleBit(const Index: TNativeUIntBitIndex) : NativeUInt`

Visibility: public

Description: `ToggleBit` reads bit value from bit number determined by argument `Index`, inverts it (if it was 0 it becomes 1, and vice versa), and stores it back.

See also: `TNativeUIntHelper.SetBit` (1918), `TNativeUIntHelper.ToggleBit` (1918), `TNativeUIntHelper.TestBit` (1919), `TNativeUIntHelper.HighestSetBitPos` (1919), `TNativeUIntHelper.LowestSetBitPos` (1919), `TNativeUIntHelper.SetBitsCount` (1920), `TNativeUIntHelper.Bits` (1920)

76.107.17 TNativeUIntHelper.TestBit

Synopsis: Check bit.

Declaration: `function TestBit (const Index: TNativeUIntBitIndex) : Boolean`

Visibility: public

Description: `TestBit` reads boolean value (true if bit is 1, and false if bit is 0) from bit number determined by argument `Index`.

See also: `TNativeUIntHelper.SetBit` (1918), `TNativeUIntHelper.ToggleBit` (1918), `TNativeUIntHelper.TestBit` (1919), `TNativeUIntHelper.HighestSetBitPos` (1919), `TNativeUIntHelper.LowestSetBitPos` (1919), `TNativeUIntHelper.SetBitsCount` (1920), `TNativeUIntHelper.Bits` (1920)

76.107.18 TNativeUIntHelper.Clear

Synopsis: Set value to 0.

Declaration: `procedure Clear`

Visibility: public

76.107.19 TNativeUIntHelper.HighestSetBitPos

Synopsis: Return the position of the leftmost bit set.

Declaration: `function HighestSetBitPos : Int8`

Visibility: public

Description: `HighestSetBitPos` scans the `NativeUInt`, starting at the leftmost position and working towards position 0, and returns the index of the first set bit. The position is measured from the 0-th, rightmost bit. When there are no bits set, the result is -1.

See also: `System.BsrByte` (1609), `TNativeUIntHelper.SetBit` (1918), `TNativeUIntHelper.ClearBit` (1918), `TNativeUIntHelper.ToggleBit` (1918), `TNativeUIntHelper.TestBit` (1919), `TNativeUIntHelper.LowestSetBitPos` (1919), `TNativeUIntHelper.SetBitsCount` (1920), `TNativeUIntHelper.Bits` (1920)

76.107.20 TNativeUIntHelper.LowestSetBitPos

Synopsis: Return the position of the rightmost bit set.

Declaration: `function LowestSetBitPos : Int8`

Visibility: public

Description: `LowestSetBitPos` scans the `NativeUInt`, starting at position 0 (rightmost position) and returns the index of the first set bit. The position is measured from the 0-th, rightmost bit. When there are no bits set, the result is -1.

See also: `System.BsfByte` (1609), `TNativeUIntHelper.SetBit` (1918), `TNativeUIntHelper.ClearBit` (1918), `TNativeUIntHelper.ToggleBit` (1918), `TNativeUIntHelper.TestBit` (1919), `TNativeUIntHelper.HighestSetBitPos` (1919), `TNativeUIntHelper.SetBitsCount` (1920), `TNativeUIntHelper.Bits` (1920)

76.107.21 TNativeUIntHelper.SetBitsCount

Synopsis: Count number of bits set.

Declaration: `function SetBitsCount : Byte`

Visibility: `public`

Description: `SetBitsCount` returns the number of bits set in a `NativeUInt`.

See also: `System.PopCnt` ([1609](#)), `TNativeUIntHelper.SetBit` ([1918](#)), `TNativeUIntHelper.ClearBit` ([1918](#)), `TNativeUIntHelper.ToggleBit` ([1918](#)), `TNativeUIntHelper.TestBit` ([1919](#)), `TNativeUIntHelper.HighestSetBitPos` ([1919](#)), `TNativeUIntHelper.LowestSetBitPos` ([1919](#)), `TNativeUIntHelper.Bits` ([1920](#))

76.107.22 TNativeUIntHelper.Bits

Synopsis: Read or write a bit.

Declaration: `Property Bits[aIndex: TNativeUIntBitIndex]: Boolean`

Visibility: `public`

Access: `Read,Write`

Description: `Bits` property reads or writes boolean value to a bit number determined by argument `aIndex`.

```
MyNativeUInt.Clear; // MyNativeUInt equals 0
MyNativeUInt.Bits[0] := true; // MyNativeUInt equals 1
MyNativeUInt.Bits[2] := true; // MyNativeUInt equals 5
WriteLn(MyNativeUInt.Bits[2]); // TRUE
```

See also: `TNativeUIntHelper.SetBit` ([1918](#)), `TNativeUIntHelper.ClearBit` ([1918](#)), `TNativeUIntHelper.ToggleBit` ([1918](#)), `TNativeUIntHelper.TestBit` ([1919](#)), `TNativeUIntHelper.HighestSetBitPos` ([1919](#)), `TNativeUIntHelper.LowestSetBitPos` ([1919](#)), `TNativeUIntHelper.SetBitsCount` ([1920](#)), `TNativeUIntHelper.Nibbles` ([1920](#)), `TNativeUIntHelper.Bytes` ([1921](#)), `TNativeUIntHelper.Words` ([1915](#)), `TNativeUIntHelper.DWords` ([1915](#))

76.107.23 TNativeUIntHelper.Nibbles

Synopsis: Read or write a nibble.

Declaration: `Property Nibbles[aIndex: TNativeUIntNibbleIndex]: nibble`

Visibility: `public`

Access: `Read,Write`

Description: `Nibbles` property reads or writes nibble value to a nibble number determined by argument `aIndex`.

```
MyNativeUInt.Clear; // MyNativeUInt equals 0
MyNativeUInt.Nibbles[1] := 3; // MyNativeUInt equals 48
WriteLn(MyNativeUInt.Nibbles[1]); // 3
```

See also: `TNativeUIntHelper.Bits` ([1920](#)), `TNativeUIntHelper.Bytes` ([1921](#)), `TNativeUIntHelper.Words` ([1915](#)), `TNativeUIntHelper.DWords` ([1915](#))

76.107.24 TNativeUIntHelper.Bytes

Synopsis: Read or write a byte.

Declaration: `Property Bytes[aIndex: TNativeUIntByteIndex]: Byte`

Visibility: `public`

Access: Read, Write

Description: `Bytes` property reads or writes byte value to a byte number determined by argument `aIndex`.

```
MyNativeUInt.Clear; // MyNativeUInt equals 0
MyNativeUInt.Bytes[1] := 1; // MyNativeUInt equals 256
WriteLn(MyNativeUInt.Bytes[1]); // 1
```

See also: `TNativeUIntHelper.Bits` ([1920](#)), `TNativeUIntHelper.Nibbles` ([1920](#)), `TNativeUIntHelper.Words` ([1915](#)), `TNativeUIntHelper.DWords` ([1915](#))

76.108 TQWordHelper**76.108.1 Description**

`TQWordHelper` contains some auxiliary routines for a `QWord`-typed ordinal value. It consists mainly of conversion routines to and from other types.

See also: `TStringHelper` ([1609](#)), `TShortIntHelper` ([1928](#)), `TSmallIntHelper` ([1955](#)), `TWordHelper` ([2000](#)), `TCardinalHelper` ([1857](#)), `TIntegerHelper` ([1893](#)), `TInt64Helper` ([1886](#)), `TByteHelper` ([1851](#)), `TNativeIntHelper` ([1908](#)), `TNativeUIntHelper` ([1915](#))

76.108.2 Method overview

Page	Method	Description
1925	<code>Clear</code>	Set value to 0.
1925	<code>ClearBit</code>	Set bit to 0.
1926	<code>HighestSetBitPos</code>	Return the position of the leftmost bit set.
1926	<code>LowestSetBitPos</code>	Return the position of the rightmost bit set.
1922	<code>Parse</code>	Convert from a string.
1924	<code>SetBit</code>	Set bit to 1.
1926	<code>SetBitsCount</code>	Count number of bits set.
1922	<code>Size</code>	Size, in bytes, of the <code>QWord</code> value.
1925	<code>TestBit</code>	Check bit.
1924	<code>ToBinString</code>	Convert to a binary string representation.
1923	<code>ToBoolean</code>	Convert to a boolean value.
1923	<code>ToDouble</code>	Convert to a double-sized floating point value.
1923	<code>ToExtended</code>	Convert to an extended-sized floating point value.
1925	<code>ToggleBit</code>	Invert bit.
1924	<code>ToHexString</code>	Convert to a hexadecimal string representation.
1924	<code>ToSingle</code>	Convert to a single-sized floating point value.
1922	<code>ToString</code>	Convert the value to string.
1923	<code>TryParse</code>	Try to convert a string to a <code>QWord</code> , report success or failure.

76.108.3 Property overview

Page	Properties	Access	Description
1926	Bits	rw	Read or write a bit.
1927	Bytes	rw	Read or write a byte.
1928	DWords	rw	Read or write a dword.
1927	Nibbles	rw	Read or write a nibble.
1927	Words	rw	Read or write a word.

76.108.4 TQWordHelper.Parse

Synopsis: Convert from a string.

Declaration: `class function Parse(const AString: string) : QWord; Static`

Visibility: public

Description: `Parse` will attempt to convert the string `AString` to a `QWord` value. It uses the `StrToQWord` ([1776](#)) function to perform the conversion, so no localization is taken into account.

Errors: If the string does not contain a valid `QWord` value, an `EConvertError` ([1812](#)) exception is raised.

See also: `TQWordHelper.ToString` ([1922](#)), `TQWordHelper.TryParse` ([1923](#)), `StrToQWord` ([1776](#))

76.108.5 TQWordHelper.Size

Synopsis: Size, in bytes, of the `QWord` value.

Declaration: `class function Size : Integer; Static`

Visibility: public

Description: `Size` returns the size (in `QWords`) of the `QWord` value. This is equivalent to `SizeOf(QWord)`.

Errors: None.

See also: `SizeOf` ([1546](#))

76.108.6 TQWordHelper.ToString

Synopsis: Convert the value to string.

Declaration: `class function ToString(const AValue: QWord) : string; Overload`
`; Static`
`function ToString : string; Overload`

Visibility: public

Description: `ToString` will, in the class function variant of this method, convert `AValue` to a string representation. In the regular method overloaded version of `ToString`, the `QWord` value itself is used. The `IntToStr` ([1738](#)) function is used to do the conversion.

See also: `TQWordHelper.Parse` ([1922](#)), `IntToStr` ([1738](#))

76.108.7 TQWordHelper.TryParse

Synopsis: Try to convert a string to a QWord, report success or failure.

Declaration: `class function TryParse(const AString: string; out AValue: QWord)
: Boolean; Static`

Visibility: public

Description: `TryParse` attempts to convert the string `AString` to a `QWord`, and reports the success of the attempt. If the attempt is successful, then `True` is returned, and the actual value of the `QWord` is returned in `AValue`.

It uses the `val` (1609) function to perform the conversion, so no localization is taken into account.

See also: `TQWordHelper.Parse` (1922), `Val` (1570)

76.108.8 TQWordHelper.ToBoolean

Synopsis: Convert to a boolean value.

Declaration: `function ToBoolean : Boolean`

Visibility: public

Description: `ToBoolean` converts the `QWord` value to a boolean: it returns `True` if the value is nonzero, `False` if it is zero.

See also: `TQWordHelper.ToSingle` (1924), `TQWordHelper.ToDouble` (1923), `TQWordHelper.ToExtended` (1923), `TQWordHelper.ToString` (1922), `TQWordHelper.ToHexString` (1924)

76.108.9 TQWordHelper.ToDouble

Synopsis: Convert to a double-sized floating point value.

Declaration: `function ToDouble : Double`

Visibility: public

Description: `ToDouble` converts the `QWord` value to a double-sized floating point value.

See also: `TQWordHelper.ToBoolean` (1923), `TQWordHelper.ToExtended` (1923), `TQWordHelper.ToSingle` (1924), `TQWordHelper.ToString` (1922), `TQWordHelper.ToHexString` (1924)

76.108.10 TQWordHelper.ToExtended

Synopsis: Convert to an extended-sized floating point value.

Declaration: `function ToExtended : Extended`

Visibility: public

Description: `ToDouble` converts the `QWord` value to an extended-sized floating point value.

See also: `TQWordHelper.ToBoolean` (1923), `TQWordHelper.ToSingle` (1924), `TQWordHelper.ToDouble` (1923), `TQWordHelper.ToString` (1922), `TQWordHelper.ToHexString` (1924)

76.108.11 TQWordHelper.ToBinString

Synopsis: Convert to a binary string representation.

Declaration: `function ToBinString : string`

Visibility: public

Description: `ToBinString` converts the `QWord` value to a binary string representation, showing all 64 bits.

See also: `TQWordHelper.ToHexString` ([1924](#)), `TQWordHelper.ToString` ([1922](#))

76.108.12 TQWordHelper.ToHexString

Synopsis: Convert to a hexadecimal string representation.

Declaration: `function ToHexString(const AMinDigits: Integer) : string; Overload`
`function ToHexString : string; Overload`

Visibility: public

Description: `ToHexString` converts the `QWord` value to a hexadecimal string representation. The `AMinDigits` argument specifies the minimal number of characters in the resulting string. The string will be left-padded with zeroes if the representation contains less than `AMinDigits` characters.

See also: `TQWordHelper.ToBoolean` ([1923](#)), `TQWordHelper.ToSingle` ([1924](#)), `TQWordHelper.ToDouble` ([1923](#)), `TQWordHelper.ToString` ([1922](#)), `TQWordHelper.ToExtended` ([1923](#))

76.108.13 TQWordHelper.ToSingle

Synopsis: Convert to an single-sized floating point value.

Declaration: `function ToSingle : Single`

Visibility: public

Description: `ToSingle` converts the `QWord` value to a single-sized floating point value.

See also: `TQWordHelper.ToBoolean` ([1923](#)), `TQWordHelper.ToDouble` ([1923](#)), `TQWordHelper.ToExtended` ([1923](#)), `TQWordHelper.ToString` ([1922](#)), `TQWordHelper.ToHexString` ([1924](#))

76.108.14 TQWordHelper.SetBit

Synopsis: Set bit to 1.

Declaration: `function SetBit(const Index: TQwordBitIndex) : QWord`

Visibility: public

Description: `SetBit` puts value 1 to bit number determined by argument `Index`.

See also: `TQWordHelper.ClearBit` ([1925](#)), `TQWordHelper.ToggleBit` ([1925](#)), `TQWordHelper.TestBit` ([1925](#)), `TQWordHelper.HighestSetBitPos` ([1926](#)), `TQWordHelper.LowestSetBitPos` ([1926](#)), `TQWordHelper.SetBitsCount` ([1926](#)), `TQWordHelper.Bits` ([1926](#))

76.108.15 TQWordHelper.ClearBit

Synopsis: Set bit to 0.

Declaration: `function ClearBit(const Index: TQwordBitIndex) : QWord`

Visibility: public

Description: `ClearBit` puts value 0 to bit number determined by argument `Index`.

See also: `TQWordHelper.SetBit` ([1924](#)), `TQWordHelper.ToggleBit` ([1925](#)), `TQWordHelper.TestBit` ([1925](#)), `TQWordHelper.HighestSetBitPos` ([1926](#)), `TQWordHelper.LowestSetBitPos` ([1926](#)), `TQWordHelper.SetBitsCount` ([1926](#)), `TQWordHelper.Bits` ([1926](#))

76.108.16 TQWordHelper.ToggleBit

Synopsis: Invert bit.

Declaration: `function ToggleBit(const Index: TQwordBitIndex) : QWord`

Visibility: public

Description: `ToggleBit` reads bit value from bit number determined by argument `Index`, inverts it (if it was 0 it becomes 1, and vice versa), and stores it back.

See also: `TQWordHelper.SetBit` ([1924](#)), `TQWordHelper.ToggleBit` ([1925](#)), `TQWordHelper.TestBit` ([1925](#)), `TQWordHelper.HighestSetBitPos` ([1926](#)), `TQWordHelper.LowestSetBitPos` ([1926](#)), `TQWordHelper.SetBitsCount` ([1926](#)), `TQWordHelper.Bits` ([1926](#))

76.108.17 TQWordHelper.TestBit

Synopsis: Check bit.

Declaration: `function TestBit(const Index: TQwordBitIndex) : Boolean`

Visibility: public

Description: `TestBit` reads boolean value (true if bit is 1, and false if bit is 0) from bit number determined by argument `Index`.

See also: `TQWordHelper.SetBit` ([1924](#)), `TQWordHelper.ToggleBit` ([1925](#)), `TQWordHelper.TestBit` ([1925](#)), `TQWordHelper.HighestSetBitPos` ([1926](#)), `TQWordHelper.LowestSetBitPos` ([1926](#)), `TQWordHelper.SetBitsCount` ([1926](#)), `TQWordHelper.Bits` ([1926](#))

76.108.18 TQWordHelper.Clear

Synopsis: Set value to 0.

Declaration: `procedure Clear`

Visibility: public

76.108.19 TQWordHelper.HighestSetBitPos

Synopsis: Return the position of the leftmost bit set.

Declaration: `function HighestSetBitPos : Int8`

Visibility: public

Description: `HighestSetBitPos` scans the `QWord`, starting at the leftmost position and working towards position 0, and returns the index of the first set bit. The position is measured from the 0-th, rightmost bit. When there are no bits set, the result is -1.

See also: `System.BsrByte` (1609), `TQWordHelper.SetBit` (1924), `TQWordHelper.ClearBit` (1925), `TQWordHelper.ToggleBit` (1925), `TQWordHelper.TestBit` (1925), `TQWordHelper.LowestSetBitPos` (1926), `TQWordHelper.SetBitsCount` (1926), `TQWordHelper.Bits` (1926)

76.108.20 TQWordHelper.LowestSetBitPos

Synopsis: Return the position of the rightmost bit set.

Declaration: `function LowestSetBitPos : Int8`

Visibility: public

Description: `LowestSetBitPos` scans the `QWord`, starting at position 0 (rightmost position) and returns the index of the first set bit. The position is measured from the 0-th, rightmost bit. When there are no bits set, the result is -1.

See also: `System.BsfByte` (1609), `TQWordHelper.SetBit` (1924), `TQWordHelper.ClearBit` (1925), `TQWordHelper.ToggleBit` (1925), `TQWordHelper.TestBit` (1925), `TQWordHelper.HighestSetBitPos` (1926), `TQWordHelper.SetBitsCount` (1926), `TQWordHelper.Bits` (1926)

76.108.21 TQWordHelper.SetBitsCount

Synopsis: Count number of bits set.

Declaration: `function SetBitsCount : Byte`

Visibility: public

Description: `SetBitsCount` returns the number of bits set in a `QWord`.

See also: `System.PopCnt` (1609), `TQWordHelper.SetBit` (1924), `TQWordHelper.ClearBit` (1925), `TQWordHelper.ToggleBit` (1925), `TQWordHelper.TestBit` (1925), `TQWordHelper.HighestSetBitPos` (1926), `TQWordHelper.LowestSetBitPos` (1926), `TQWordHelper.Bits` (1926)

76.108.22 TQWordHelper.Bits

Synopsis: Read or write a bit.

Declaration: `Property Bits[aIndex: TQwordBitIndex]: Boolean`

Visibility: public

Access: Read,Write

Description: `Bits` property reads or writes boolean value to a bit number determined by argument `aIndex`.

```
MyQWord.Clear; // %000000000000000000000000000000000000000000000 M  
    MyQWord.Bits[0] := true;   // %0000000000000000000000000000000000000000  
    MyQWord.Bits[2] := true;   // %0000000000000000000000000000000000000000  
WriteLn(MyQWord.Bits[2]); // TRUE
```

See also: [TQWordHelper.SetBit \(1924\)](#), [TQWordHelper.ClearBit \(1925\)](#), [TQWordHelper.ToggleBit \(1925\)](#), [TQWordHelper.TestBit \(1925\)](#), [TQWordHelper.HighestSetBitPos \(1926\)](#), [TQWordHelper.LowestSetBitPos \(1926\)](#), [TQWordHelper.SetBitsCount \(1926\)](#), [TQWordHelper.Nibbles \(1927\)](#), [TQWordHelper.Bytes \(1927\)](#), [TQWordHelper.Words \(1927\)](#), [TQWordHelper.DWords \(1928\)](#)

76.108.23 TQWordHelper.Nibbles

Synopsis: Read or write a nibble.

```
Declaration: Property Nibbles[aIndex: TQwordNibbleIndex]: nibble
```

Visibility: public

Access: Read, Write

Description: `Nibbles` property reads or writes nibble value to a nibble number determined by argument `aIndex`.

[illegible]

See also: TQWordHelper.Bits (1926), TQWordHelper.Bytes (1927), TQWordHelper.Words (1927), TQWordHelper.DWords (1928)

76.108.24 TQWordHelper.Bytes

Synopsis: Read or write a byte.

Declaration: Property Bytes[aIndex: TQwordByteIndex]: Byte

Visibility: public

Access: Read, Write

Description: Bytes property reads or writes byte value to a byte number determined by argument aIndex.

[illegible]

See also: TQWordHelper.Bits (1926), TQWordHelper.Nibbles (1927), TQWordHelper.Words (1927), TQWordHelper.DWords (1928)

76.108.25 TQWordHelper.Words

Synopsis: Read or write a word.

```
Declaration: Property Words[aIndex: TQwordWordIndex]: Word
```

Visibility: public

Access: Read, Write

Description: Words property reads or writes word value to a word number determined by argument aIndex.

[illegible]

See also: [TQWordHelper.Bits \(1926\)](#), [TQWordHelper.Nibbles \(1927\)](#), [TQWordHelper.Bytes \(1927\)](#), [TQWordHelper.DWords \(1928\)](#)

76.108.26 TQWordHelper.DWords

Synopsis: Read or write a dword.

Declaration: Property DWords[aIndex: T0wordDwordIndex]: DWord

Visibility: public

Access: Read, Write

Description: DWords property reads or writes dword value to a dword number determined by argument aIndex.

[illegible]

See also: TQWordHelper.Bits (1926), TQWordHelper.Nibbles (1927), TQWordHelper.Bytes (1927), TQWordHelper.Words (1927)

76.109 TShortIntHelper

76.109.1 Description

`TShortIntHelper` contains some auxiliary routines for a `ShortInt`-typed ordinal value. It consists mainly of conversion routines to and from other types.

See also: TStringHelper ([1609](#)), TByteHelper ([1851](#)), TSmallIntHelper ([1955](#)), TWordHelper ([2000](#)), TCardinalHelper ([1857](#)), TIntegerHelper ([1893](#)), TInt64Helper ([1886](#)), TQWordHelper ([1921](#)), TNativeIntHelper ([1908](#)), TNativeUIntHelper ([1915](#))

76.109.2 Method overview

Page	Method	Description
1933	Clear	Set value to 0.
1932	ClearBit	Set bit to 0.
1933	HighestSetBitPos	Return the position of the leftmost bit set.
1933	LowestSetBitPos	Return the position of the rightmost bit set.
1929	Parse	Convert from a string.
1932	SetBit	Set bit to 1.
1933	SetBitsCount	Count number of bits set.
1929	Size	Size, in bytes, of the ShortInt value.
1932	TestBit	Check bit.
1931	ToBinString	Convert to a binary string representation.
1930	ToBoolean	Convert to a boolean value.
1930	ToDouble	Convert to a double-sized floating point value.
1931	ToExtended	Convert to an extended-sized floating point value.
1932	ToggleBit	Invert bit.
1931	ToHexString	Convert to a hexadecimal string representation.
1931	ToSingle	Convert to a single-sized floating point value.
1930	ToString	Convert the value to string.
1930	TryParse	Try to convert a string to a ShortInt, report success or failure.

76.109.3 Property overview

Page	Properties	Access	Description
1934	Bits	rw	Read or write a bit.
1934	Nibbles	rw	Read or write a nibble.

76.109.4 TShortIntHelper.Parse

Synopsis: Convert from a string.

Declaration: `class function Parse(const AString: string) : ShortInt; Static`

Visibility: public

Description: `Parse` will attempt to convert the string `AString` to a `ShortInt` value. It uses the `StrToInt` ([1773](#)) function to perform the conversion, so no localization is taken into account.

Errors: If the string does not contain a valid `ShortInt` value, an `EConvertError` ([1812](#)) exception is raised.

See also: `TShortIntHelper.ToString` ([1930](#)), `TShortIntHelper.TryParse` ([1930](#)), `StrToInt` ([1773](#))

76.109.5 TShortIntHelper.Size

Synopsis: Size, in bytes, of the `ShortInt` value.

Declaration: `class function Size : Integer; Static`

Visibility: public

Description: `Size` returns the size (in `ShortInt`s) of the `ShortInt` value. This is equivalent to `SizeOf(Byte)`.

Errors: None.

See also: `SizeOf` ([1546](#))

76.109.6 TShortIntHelper.ToString

Synopsis: Convert the value to string.

```
Declaration: class function ToString(const AValue: ShortInt) : string; Overload
                ; Static
                function ToString : string; Overload
```

Visibility: public

Description: ToString will, in the class function variant of this method, convert AValue to a string representation. In the regular method overloaded version of ToString, the ShortInt value itself is used. The IntToStr (1738) function is used to do the conversion.

See also: [TShortIntHelper.Parse \(1929\)](#), [IntToStr \(1738\)](#)

76.109.7 TShortIntHelper.TryParse

Synopsis: Try to convert a string to a `ShortInt`, report success or failure.

```
Declaration: class function TryParse(const AString: string; out AValue: ShortInt)
                                     : Boolean; Static
```

Visibility: public

Description: TryParse attempts to convert the string AString to a ShortInt, and reports the success of the attempt. If the attempt is successful, then True is returned, and the actual value of the ShortInt is returned in AValue.

It uses the `val (1609)` function to perform the conversion, so no localization is taken into account.

See also: [TShortIntHelper.Parse \(1929\)](#), [Val \(1570\)](#)

76.109.8 TShortIntHelper.ToBoolean

Synopsis: Convert to a boolean value.

```
Declaration: function ToBoolean : Boolean
```

Visibility: public

Description: `ToBoolean` converts the `ShortInt` value to a boolean: it returns `True` if the value is nonzero, `False` if it is zero.

See also: [TShortIntHelper.ToSingle \(1931\)](#), [TShortIntHelper.ToDouble \(1930\)](#), [TShortIntHelper.ToExtended \(1931\)](#), [TShortIntHelper.ToString \(1930\)](#), [TShortIntHelper.ToHexString \(1931\)](#)

76.109.9 TShortIntHelper.ToDouble

Synopsis: Convert to a double-sized floating point value.

```
Declaration: function ToDouble : Double
```

Visibility: public

Description: ToDouble converts the ShortInt value to a double-sized floating point value.

See also: [TShortIntHelper.ToBoolean \(1930\)](#), [TShortIntHelper.ToExtended \(1931\)](#), [TShortIntHelper.ToSingle \(1931\)](#), [TShortIntHelper.ToString \(1930\)](#), [TShortIntHelper.ToHexString \(1931\)](#)

76.109.10 TShortIntHelper.ToExtended

Synopsis: Convert to an extended-sized floating point value.

Declaration: `function ToExtended : Extended`

Visibility: public

Description: `ToDouble` converts the `ShortInt` value to an extended-sized floating point value.

See also: `TShortIntHelper.ToBoolean` ([1930](#)), `TShortIntHelper.ToSingle` ([1931](#)), `TShortIntHelper.ToDouble` ([1930](#)), `TShortIntHelper.ToString` ([1930](#)), `TShortIntHelper.ToHexString` ([1931](#))

76.109.11 TShortIntHelper.ToBinString

Synopsis: Convert to a binary string representation.

Declaration: `function ToBinString : string`

Visibility: public

Description: `ToBinString` converts the `shortint` value to a binary string representation, showing all 8 bits.

See also: `TShortIntHelper.ToHexString` ([1931](#)), `TShortIntHelper.ToString` ([1930](#))

76.109.12 TShortIntHelper.ToHexString

Synopsis: Convert to a hexadecimal string representation.

Declaration: `function ToHexString(const AMinDigits: Integer) : string; Overload`
`function ToHexString : string; Overload`

Visibility: public

Description: `ToHexString` converts the `ShortInt` value to a hexadecimal string representation. The `AMinDigits` argument specifies the minimal number of characters in the resulting string. The string will be left-padded with zeroes if the representation contains less than `AMinDigits` characters.

See also: `TShortIntHelper.ToBoolean` ([1930](#)), `TShortIntHelper.ToSingle` ([1931](#)), `TShortIntHelper.ToDouble` ([1930](#)), `TShortIntHelper.ToString` ([1930](#)), `TShortIntHelper.ToExtended` ([1931](#))

76.109.13 TShortIntHelper.ToSingle

Synopsis: Convert to an single-sized floating point value.

Declaration: `function ToSingle : Single`

Visibility: public

Description: `ToSingle` converts the `ShortInt` value to a single-sized floating point value.

See also: `TShortIntHelper.ToBoolean` ([1930](#)), `TShortIntHelper.ToDouble` ([1930](#)), `TShortIntHelper.ToExtended` ([1931](#)), `TShortIntHelper.ToString` ([1930](#)), `TShortIntHelper.ToHexString` ([1931](#))

76.109.14 TShortIntHelper.SetBit

Synopsis: Set bit to 1.

Declaration: `function SetBit(const Index: TShortIntBitIndex) : ShortInt`

Visibility: public

Description: `SetBit` puts value 1 to bit number determined by argument `Index`.

See also: `TShortIntHelper.ClearBit` ([1932](#)), `TShortIntHelper.ToggleBit` ([1932](#)), `TShortIntHelper.TestBit` ([1932](#)), `TShortIntHelper.HighestSetBitPos` ([1933](#)), `TShortIntHelper.LowestSetBitPos` ([1933](#)), `TShortIntHelper.SetBitsCount` ([1933](#)), `TShortIntHelper.Bits` ([1934](#))

76.109.15 TShortIntHelper.ClearBit

Synopsis: Set bit to 0.

Declaration: `function ClearBit(const Index: TShortIntBitIndex) : ShortInt`

Visibility: public

Description: `ClearBit` puts value 0 to bit number determined by argument `Index`.

See also: `TShortIntHelper.SetBit` ([1932](#)), `TShortIntHelper.ToggleBit` ([1932](#)), `TShortIntHelper.TestBit` ([1932](#)), `TShortIntHelper.HighestSetBitPos` ([1933](#)), `TShortIntHelper.LowestSetBitPos` ([1933](#)), `TShortIntHelper.SetBitsCount` ([1933](#)), `TShortIntHelper.Bits` ([1934](#))

76.109.16 TShortIntHelper.ToggleBit

Synopsis: Invert bit.

Declaration: `function ToggleBit(const Index: TShortIntBitIndex) : ShortInt`

Visibility: public

Description: `ToggleBit` reads bit value from bit number determined by argument `Index`, inverts it (if it was 0 it becomes 1, and vice versa), and stores it back.

See also: `TShortIntHelper.SetBit` ([1932](#)), `TShortIntHelper.ClearBit` ([1932](#)), `TShortIntHelper.TestBit` ([1932](#)), `TShortIntHelper.HighestSetBitPos` ([1933](#)), `TShortIntHelper.LowestSetBitPos` ([1933](#)), `TShortIntHelper.SetBitsCount` ([1933](#)), `TShortIntHelper.Bits` ([1934](#))

76.109.17 TShortIntHelper.TestBit

Synopsis: Check bit.

Declaration: `function TestBit(const Index: TShortIntBitIndex) : Boolean`

Visibility: public

Description: `TestBit` reads boolean value (true if bit is 1, and false if bit is 0) from bit number determined by argument `Index`.

See also: `TShortIntHelper.SetBit` ([1932](#)), `TShortIntHelper.ClearBit` ([1932](#)), `TShortIntHelper.ToggleBit` ([1932](#)), `TShortIntHelper.HighestSetBitPos` ([1933](#)), `TShortIntHelper.LowestSetBitPos` ([1933](#)), `TShortIntHelper.SetBitsCount` ([1933](#)), `TShortIntHelper.Bits` ([1934](#))

76.109.18 TShortIntHelper.Clear

Synopsis: Set value to 0.

Declaration: `procedure Clear`

Visibility: `public`

76.109.19 TShortIntHelper.HighestSetBitPos

Synopsis: Return the position of the leftmost bit set.

Declaration: `function HighestSetBitPos : Int8`

Visibility: `public`

Description: `HighestSetBitPos` scans the shortint, starting at the leftmost position and working towards position 0, and returns the index of the first set bit. The position is measured from the 0-th, rightmost bit. When there are no bits set, the result is -1.

See also: `System.BsrByte` ([1609](#)), `TShortIntHelper.SetBit` ([1932](#)), `TShortIntHelper.ClearBit` ([1932](#)), `TShortIntHelper.ToggleBit` ([1932](#)), `TShortIntHelper.TestBit` ([1932](#)), `TShortIntHelper.LowestSetBitPos` ([1933](#)), `TShortIntHelper.SetBitsCount` ([1933](#)), `TShortIntHelper.Bits` ([1934](#))

76.109.20 TShortIntHelper.LowestSetBitPos

Synopsis: Return the position of the rightmost bit set.

Declaration: `function LowestSetBitPos : Int8`

Visibility: `public`

Description: `LowestSetBitPos` scans the shortint, starting at position 0 (rightmost position) and returns the index of the first set bit. The position is measured from the 0-th, rightmost bit. When there are no bits set, the result is -1.

See also: `System.BsfByte` ([1609](#)), `TShortIntHelper.SetBit` ([1932](#)), `TShortIntHelper.ClearBit` ([1932](#)), `TShortIntHelper.ToggleBit` ([1932](#)), `TShortIntHelper.TestBit` ([1932](#)), `TShortIntHelper.HighestSetBitPos` ([1933](#)), `TShortIntHelper.SetBitsCount` ([1933](#)), `TShortIntHelper.Bits` ([1934](#))

76.109.21 TShortIntHelper.SetBitsCount

Synopsis: Count number of bits set.

Declaration: `function SetBitsCount : Byte`

Visibility: `public`

Description: `SetBitsCount` returns the number of bits set in a shortint.

See also: `System.PopCnt` ([1609](#)), `TShortIntHelper.SetBit` ([1932](#)), `TShortIntHelper.ClearBit` ([1932](#)), `TShortIntHelper.ToggleBit` ([1932](#)), `TShortIntHelper.TestBit` ([1932](#)), `TShortIntHelper.HighestSetBitPos` ([1933](#)), `TShortIntHelper.LowestSetBitPos` ([1933](#)), `TShortIntHelper.Bits` ([1934](#))

76.109.22 TShortIntHelper.Bits

Synopsis: Read or write a bit.

Declaration: `Property Bits[aIndex: TShortIntBitIndex]: Boolean`

Visibility: public

Access: Read,Write

Description: `Bits` property reads or writes boolean value to a bit number determined by argument `aIndex`.

```
MyShortInt.Clear; // %000000000 MyShortInt equals 0
MyShortInt.Bits[0] := true; // %000000001 MyShortInt equals 1
MyShortInt.Bits[2] := true; // %00000101 MyShortInt equals 5
WriteLn(MyShortInt.Bits[2]); // TRUE
```

See also: `TShortIntHelper.SetBit` ([1932](#)), `TShortIntHelper.ClearBit` ([1932](#)), `TShortIntHelper.ToggleBit` ([1932](#)), `TShortIntHelper.TestBit` ([1932](#)), `TShortIntHelper.HighestSetBitPos` ([1933](#)), `TShortIntHelper.LowestSetBitPos` ([1933](#)), `TShortIntHelper.SetBitsCount` ([1933](#)), `TShortIntHelper.Nibbles` ([1934](#))

76.109.23 TShortIntHelper.Nibbles

Synopsis: Read or write a nibble.

Declaration: `Property Nibbles[aIndex: TShortIntNibbleIndex]: nibble`

Visibility: public

Access: Read,Write

Description: `Nibbles` property reads or writes nibble value to a nibble number determined by argument `aIndex`.

```
MyShortInt.Clear; // %000000000 MyShortInt equals 0
MyShortInt.Nibbles[1] := 3; // %00110000 MyShortInt equals 48
WriteLn(MyShortInt.Nibbles[1]); // 3
```

See also: `TShortIntHelper.Bits` ([1934](#))

76.110 TShortStringHelper

76.110.1 Method overview

Page	Method	Description
1937	Compare	
1937	CompareOrdinal	
1937	CompareText	
1940	CompareTo	
1940	Contains	
1937	Copy	
1941	CopyTo	
1941	CountChar	
1938	Create	
1941	DeQuotedString	
1938	EndsText	
1941	EndsWith	
1938	Equals	
1938	Format	
1941	GetHashCode	
1941	IndexOf	
1942	IndexOfAny	
1942	IndexOfAnyUnquoted	
1942	IndexOfUnquoted	
1942	Insert	
1943	IsDelimiter	
1943	IsEmpty	
1938	IsNullOrEmpty	
1938	IsNullOrWhiteSpace	
1939	Join	
1943	LastDelimiter	
1943	LastIndexOf	
1943	LastIndexOfAny	
1939	LowerCase	
1943	PadLeft	
1944	PadRight	
1939	Parse	
1944	QuotedString	
1944	Remove	
1944	Replace	
1944	Split	
1945	StartsWith	
1945	Substring	
1939	ToBoolean	
1946	ToCharArray	
1939	ToDouble	
1940	ToExtended	
1940	ToInt64	
1940	ToInteger	
1946	ToLower	
1946	ToLowerInvariant	
1940	ToSingle	
1946	ToUpper	
1946	ToUpperInvariant	
1946	Trim	
1947	TrimEnd	
1946	TrimLeft	1936
1947	TrimRight	
1947	TrimStart	
1940	UpperCase	

76.110.2 Property overview

Page	Properties	Access	Description
1947	Chars	r	
1947	Length	r	

76.110.3 TShortStringHelper.Compare

```

Declaration: class function Compare(const A: ShortString; const B: ShortString)
            : Integer; Overload; Static
class function Compare(const A: ShortString; const B: ShortString;
            IgnoreCase: Boolean) : Integer; Overload; Static
class function Compare(const A: ShortString; const B: ShortString;
            Options: TCompareOptions) : Integer; Overload
            ; Static
class function Compare(const A: ShortString; IndexA: SizeInt;
            const B: ShortString; IndexB: SizeInt;
            ALen: SizeInt) : Integer; Overload; Static
class function Compare(const A: ShortString; IndexA: SizeInt;
            const B: ShortString; IndexB: SizeInt;
            ALen: SizeInt; IgnoreCase: Boolean) : Integer
            ; Overload; Static
class function Compare(const A: ShortString; IndexA: SizeInt;
            const B: ShortString; IndexB: SizeInt;
            ALen: SizeInt; Options: TCompareOptions) : Integer
            ; Overload; Static

```

Visibility: public

76.110.4 TShortStringHelper.CompareOrdinal

```

Declaration: class function CompareOrdinal(const A: ShortString;
                                           const B: ShortString) : Integer;  Overload
                                           ;  Static
class function CompareOrdinal(const A: ShortString; IndexA: SizeInt;
                               const B: ShortString; IndexB: SizeInt;
                               ALen: SizeInt) : Integer;  Overload
                               ;  Static

```

Visibility: public

76.110.5 TShortStringHelper.CompareText

```
Declaration: class function CompareText(const A: ShortString; const B: ShortString)
                                         : Integer; Static
```

Visibility: public

76.110.6 TShortStringHelper.Copy

```
Declaration: class function Copy(const Str: ShortString) : ShortString; Static
```

Visibility: public

76.110.7 TShortStringHelper.Create

```

Declaration: class function Create(AChar: AnsiChar; ACount: SizeInt) : ShortString
              ; Overload; Static
class function Create(const AValue: Array of AnsiChar) : ShortString
              ; Overload; Static
class function Create(const AValue: Array of AnsiChar;
                      StartIndex: SizeInt; ALen: SizeInt) : ShortString
              ; Overload; Static

```

Visibility: public

76.110.8 TShortStringHelper.EndsText

[illegible]

Visibility: public

76.110.9 TShortStringHelper.Equals

```
Declaration: class function Equals(const a: ShortString; const b: ShortString)
                : Boolean; Overload; Static
function Equals(const AValue: ShortString; IgnoreCase: Boolean)
                : Boolean; Overload
```

Visibility: public

76.110.10 TShortStringHelper.Format

```
Declaration: class function Format(const AFormat: ShortString;
                                const args: Array of const) : ShortString
                                ; Overload; Static
function Format(const args: Array of const) : ShortString; Overload
```

Visibility: public

76.110.11 TShortStringHelper.IsNullOrEmpty

```
Declaration: class function IsNullOrEmpty(const AValue: ShortString) : Boolean
                                         ; Static
```

Visibility: public

76.110.12 TShortStringHelper.IsNullOrWhiteSpace

```
Declaration: class function IsNullOrWhiteSpace(const AValue: ShortString) : Boolean
                                                    ; Static
```

Visibility: public

76.110.13 TShortStringHelper.Join

Declaration: class function Join(const Separator: ShortString;
 const Values: Array of const) : ShortString
 ; Overload; Static
 class function Join(const Separator: ShortString;
 const Values: Array of ShortString) : ShortString
 ; Overload; Static
 class function Join(const Separator: ShortString;
 const Values: Array of ShortString;
 StartIndex: SizeInt; ACount: SizeInt) : ShortString
 ; Overload; Static

Visibility: public

76.110.14 TShortStringHelper.LowerCase

Declaration: class function LowerCase(const S: ShortString) : ShortString; Overload
 ; Static

Visibility: public

76.110.15 TShortStringHelper.Parse

Declaration: class function Parse(const AValue: Boolean) : ShortString; Overload
 ; Static
 class function Parse(const AValue: Extended) : ShortString; Overload
 ; Static
 class function Parse(const AValue: Int64) : ShortString; Overload
 ; Static
 class function Parse(const AValue: Integer) : ShortString; Overload
 ; Static

Visibility: public

76.110.16 TShortStringHelper.ToBoolean

Declaration: class function ToBoolean(const S: ShortString) : Boolean; Overload
 ; Static
 function ToBoolean : Boolean; Overload

Visibility: public

76.110.17 TShortStringHelper.ToDouble

Declaration: class function ToDouble(const S: ShortString) : Double; Overload
 ; Static
 function ToDouble : Double; Overload

Visibility: public

76.110.18 TShortStringHelper.ToExtended

```
Declaration: class function ToExtended(const S: ShortString) : Extended; Overload
                ; Static
                function ToExtended : Extended; Overload
```

Visibility: public

76.110.19 TShortStringHelper.ToInt64

```
Declaration: class function ToInt64(const S: ShortString) : Int64; Overload; Static
            function ToInt64 : Int64; Overload
```

Visibility: public

76.110.20 TShortStringHelper.ToInteger

```
Declaration: class function ToInteger(const S: ShortString) : Integer; Overload
              ; Static
              function ToInteger : Integer; Overload
```

Visibility: public

76.110.21 TShortStringHelper.ToSingle

```
Declaration: class function ToSingle(const S: ShortString) : Single; Overload
              ; Static
              function ToSingle : Single; Overload
```

Visibility: public

76.110.22 TShortStringHelper.UpperCase

```
Declaration: class function UpperCase(const S: ShortString) : ShortString; Overload
                ; Static
```

Visibility: public

76.110.23 TShortStringHelper.CompareTo

Declaration: `function CompareTo(const B: ShortString) : Integer`

Visibility: public

76.110.24 TShortStringHelper.Contains

```
Declaration: function Contains(const AValue: ShortString; IgnoreCase: Boolean)
                : Boolean
```

Visibility: public

76.110.31 TShortStringHelper.IndexOfUnQuoted

Declaration: function IndexOfUnQuoted(const AValue: ShortString;
 StartQuote: AnsiChar; EndQuote: AnsiChar;
 StartIndex: SizeInt) : SizeInt; Overload

Visibility: public

76.110.32 TShortStringHelper.IndexOfAny

Declaration: function IndexOfAny(const AnyOf: Array of AnsiChar) : SizeInt; Overload
 function IndexOfAny(const AnyOf: Array of AnsiChar; StartIndex: SizeInt)
 : SizeInt; Overload
 function IndexOfAny(const AnyOf: Array of AnsiChar;
 StartIndex: SizeInt; ACount: SizeInt) : SizeInt
 ; Overload
 function IndexOfAny(const AnyOf: Array of ShortString) : SizeInt
 ; Overload
 function IndexOfAny(const AnyOf: Array of ShortString;
 StartIndex: SizeInt) : SizeInt; Overload
 function IndexOfAny(const AnyOf: Array of ShortString;
 StartIndex: SizeInt; ACount: SizeInt) : SizeInt
 ; Overload
 function IndexOfAny(const AnyOf: Array of ShortString;
 StartIndex: SizeInt; ACount: SizeInt;
 out AMatch: SizeInt) : SizeInt; Overload

Visibility: public

76.110.33 TShortStringHelper.IndexOfAnyUnquoted

Declaration: function IndexOfAnyUnquoted(const AnyOf: Array of AnsiChar;
 StartQuote: AnsiChar; EndQuote: AnsiChar)
 : SizeInt; Overload
 function IndexOfAnyUnquoted(const AnyOf: Array of AnsiChar;
 StartQuote: AnsiChar; EndQuote: AnsiChar;
 StartIndex: SizeInt) : SizeInt; Overload
 function IndexOfAnyUnquoted(const AnyOf: Array of AnsiChar;
 StartQuote: AnsiChar; EndQuote: AnsiChar;
 StartIndex: SizeInt; ACount: SizeInt)
 : SizeInt; Overload
 function IndexOfAnyUnquoted(const AnyOf: Array of ShortString;
 StartQuote: AnsiChar; EndQuote: AnsiChar;
 StartIndex: SizeInt; out Matched: SizeInt)
 : SizeInt; Overload

Visibility: public

76.110.34 TShortStringHelper.Insert

Declaration: function Insert(StartIndex: SizeInt; const AValue: ShortString)
 : ShortString

Visibility: public

76.110.35 TShortStringHelper.IsDelimiter

Declaration: function IsDelimiter(const Delimiters: ShortString; Index: SizeInt)
: Boolean

Visibility: public

76.110.36 TShortStringHelper.IsEmpty

Declaration: function IsEmpty : Boolean

Visibility: public

76.110.37 TShortStringHelper.LastDelimiter

Declaration: function LastDelimiter(const Delims: ShortString) : SizeInt

Visibility: public

76.110.38 TShortStringHelper.LastIndexOf

Declaration: function LastIndexOf(AValue: AnsiChar) : SizeInt; Overload
function LastIndexOf(const AValue: ShortString) : SizeInt; Overload
function LastIndexOf(AValue: AnsiChar; AStartIndex: SizeInt) : SizeInt
; Overload
function LastIndexOf(const AValue: ShortString; AStartIndex: SizeInt)
: SizeInt; Overload
function LastIndexOf(AValue: AnsiChar; AStartIndex: SizeInt;
ACount: SizeInt) : SizeInt; Overload
function LastIndexOf(const AValue: ShortString; AStartIndex: SizeInt;
ACount: SizeInt) : SizeInt; Overload

Visibility: public

76.110.39 TShortStringHelper.LastIndexOfAny

Declaration: function LastIndexOfAny(const AnyOf: Array of AnsiChar) : SizeInt
; Overload
function LastIndexOfAny(const AnyOf: Array of AnsiChar;
AStartIndex: SizeInt) : SizeInt; Overload
function LastIndexOfAny(const AnyOf: Array of AnsiChar;
AStartIndex: SizeInt; ACount: SizeInt) : SizeInt
; Overload

Visibility: public

76.110.40 TShortStringHelper.PadLeft

Declaration: function PadLeft(ATotalWidth: SizeInt) : ShortString; Overload
function PadLeft(ATotalWidth: SizeInt; PaddingChar: AnsiChar)
: ShortString; Overload

Visibility: public

76.110.41 TShortStringHelper.PadRight

Declaration: function PadRight(ATotalWidth: SizeInt) : ShortString; Overload
function PadRight(ATotalWidth: SizeInt; PaddingChar: AnsiChar)
: ShortString; Overload

Visibility: public

76.110.42 TShortStringHelper.QuotedString

Declaration: function QuotedString : ShortString; Overload
function QuotedString(const AQuoteChar: AnsiChar) : ShortString
; Overload

Visibility: public

76.110.43 TShortStringHelper.Remove

Declaration: function Remove(StartIndex: SizeInt) : ShortString; Overload
function Remove(StartIndex: SizeInt; ACount: SizeInt) : ShortString
; Overload

Visibility: public

76.110.44 TShortStringHelper.Replace

Declaration: function Replace(OldChar: AnsiChar; NewChar: AnsiChar) : ShortString
; Overload
function Replace(OldChar: AnsiChar; NewChar: AnsiChar;
ReplaceFlags: TReplaceFlags) : ShortString; Overload
function Replace(const OldValue: ShortString;
const NewValue: ShortString) : ShortString; Overload
function Replace(const OldValue: ShortString;
const NewValue: ShortString;
ReplaceFlags: TReplaceFlags) : ShortString; Overload

Visibility: public

76.110.45 TShortStringHelper.Split

Declaration: function Split(const Separators: Array of AnsiChar) : TShortStringArray
; Overload
function Split(const Separators: Array of AnsiChar; ACount: SizeInt)
: TShortStringArray; Overload
function Split(const Separators: Array of AnsiChar;
Options: TStringSplitOptions) : TShortStringArray
; Overload
function Split(const Separators: Array of AnsiChar; ACount: SizeInt;
Options: TStringSplitOptions) : TShortStringArray
; Overload
function Split(const Separators: Array of ShortString)
: TShortStringArray; Overload
function Split(const Separators: Array of ShortString; ACount: SizeInt)
: TShortStringArray; Overload

```

function Split(const Separators: Array of ShortString;
               Options: TStringSplitOptions) : TShortStringArray
    ; Overload
function Split(const Separators: Array of ShortString; ACount: SizeInt;
               Options: TStringSplitOptions) : TShortStringArray
    ; Overload
function Split(const Separators: Array of AnsiChar; AQuote: AnsiChar)
    : TShortStringArray; Overload
function Split(const Separators: Array of AnsiChar;
               AQuoteStart: AnsiChar; AQuoteEnd: AnsiChar)
    : TShortStringArray; Overload
function Split(const Separators: Array of AnsiChar;
               AQuoteStart: AnsiChar; AQuoteEnd: AnsiChar;
               Options: TStringSplitOptions) : TShortStringArray
    ; Overload
function Split(const Separators: Array of AnsiChar;
               AQuoteStart: AnsiChar; AQuoteEnd: AnsiChar;
               ACount: SizeInt) : TShortStringArray; Overload
function Split(const Separators: Array of AnsiChar;
               AQuoteStart: AnsiChar; AQuoteEnd: AnsiChar;
               ACount: SizeInt; Options: TStringSplitOptions)
    : TShortStringArray; Overload
function Split(const Separators: Array of ShortString; AQuote: AnsiChar)
    : TShortStringArray; Overload
function Split(const Separators: Array of ShortString;
               AQuoteStart: AnsiChar; AQuoteEnd: AnsiChar)
    : TShortStringArray; Overload
function Split(const Separators: Array of ShortString;
               AQuoteStart: AnsiChar; AQuoteEnd: AnsiChar;
               Options: TStringSplitOptions) : TShortStringArray
    ; Overload
function Split(const Separators: Array of ShortString;
               AQuoteStart: AnsiChar; AQuoteEnd: AnsiChar;
               ACount: SizeInt) : TShortStringArray; Overload
function Split(const Separators: Array of ShortString;
               AQuoteStart: AnsiChar; AQuoteEnd: AnsiChar;
               ACount: SizeInt; Options: TStringSplitOptions)
    : TShortStringArray; Overload

```

Visibility: public

76.110.46 TShortStringHelper.StartsWith

Declaration: function StartsWith(const AValue: ShortString) : Boolean; Overload
 function StartsWith(const AValue: ShortString; IgnoreCase: Boolean)
 : Boolean; Overload

Visibility: public

76.110.47 TShortStringHelper.Substring

Declaration: function Substring(AStartIndex: SizeInt) : ShortString; Overload
 function Substring(AStartIndex: SizeInt; ALen: SizeInt) : ShortString

; Overload

Visibility: public

76.110.48 TShortStringHelper.ToCharArray

Declaration: function ToCharArray : TCharArray; Overload
function ToCharArray(AStartIndex: SizeInt; ALen: SizeInt) : TCharArray
; Overload

Visibility: public

76.110.49 TShortStringHelper.ToLower

Declaration: function ToLower : ShortString; Overload

Visibility: public

76.110.50 TShortStringHelper.ToLowerInvariant

Declaration: function ToLowerInvariant : ShortString

Visibility: public

76.110.51 TShortStringHelper.ToUpper

Declaration: function ToUpper : ShortString; Overload

Visibility: public

76.110.52 TShortStringHelper.ToUpperInvariant

Declaration: function ToUpperInvariant : ShortString

Visibility: public

76.110.53 TShortStringHelper.Trim

Declaration: function Trim : ShortString; Overload
function Trim(const ATrimChars: Array of AnsiChar) : ShortString
; Overload

Visibility: public

76.110.54 TShortStringHelper.TrimLeft

Declaration: function TrimLeft : ShortString; Overload
function TrimLeft(const ATrimChars: Array of AnsiChar) : ShortString
; Overload

Visibility: public

76.110.55 TShortStringHelper.TrimRight

Declaration: `function TrimRight : ShortString; Overload`
`function TrimRight(const ATrimChars: Array of AnsiChar) : ShortString`
`; Overload`

Visibility: public

76.110.56 TShortStringHelper.TrimEnd

Declaration: `function TrimEnd(const ATrimChars: Array of AnsiChar) : ShortString`

Visibility: public

76.110.57 TShortStringHelper.TrimStart

Declaration: `function TrimStart(const ATrimChars: Array of AnsiChar) : ShortString`

Visibility: public

76.110.58 TShortStringHelper.Chars

Declaration: `Property Chars[AIndex: SizeInt]: AnsiChar`

Visibility: public

Access: Read

76.110.59 TShortStringHelper.Length

Declaration: `Property &Length : SizeInt`

Visibility: public

Access: Read

76.111 TSimpleRWSync

76.111.1 Description

`TSimpleRWSync` implements a simple read/write locking mechanism. It controls access to an object: only a single thread is allowed access to an object for either read or write operations.

Access is controlled through a single critical section.

See also: `TMultiReadExclusiveWriteSynchronizer` ([1906](#))

76.111.2 Interfaces overview

Page	Interfaces	Description
1825	<code>IReadWriteSync</code>	Read/Write synchronizer.

76.111.3 Method overview

Page	Method	Description
1949	Beginread	Acquire a read lock.
1948	Beginwrite	Acquire a write lock.
1948	Create	Create a new instance of <code>TSimpleRWSync</code> .
1948	Destroy	Removes the <code>TSimpleRWSync</code> instance from memory.
1949	Endread	Release the read lock.
1948	Endwrite	Release the write lock.

76.111.4 TSimpleRWSync.Create

Synopsis: Create a new instance of `TSimpleRWSync`.

Declaration: `constructor Create; Virtual`

Visibility: `public`

Description: `Create` allocates a new instance of `TSimpleRWSync` and initializes the critical section for use in the various methods.

See also: `TSimpleRWSync.Destroy` ([1948](#))

76.111.5 TSimpleRWSync.Destroy

Synopsis: Removes the `TSimpleRWSync` instance from memory.

Declaration: `destructor Destroy; Override`

Visibility: `public`

Description: `Destroy` releases the critical section and removes the `TSimpleRWSync` instance from memory.

See also: `TSimpleRWSync.Create` ([1948](#))

76.111.6 TSimpleRWSync.Beginwrite

Synopsis: Acquire a write lock.

Declaration: `function Beginwrite : Boolean`

Visibility: `public`

Description: `Beginwrite` waits till all other threads have released their read or write locks on the object, and then acquires a write lock on the object.

See also: `TSimpleRWSync.BeginRead` ([1949](#)), `TSimpleRWSync.EndWrite` ([1948](#))

76.111.7 TSimpleRWSync.Endwrite

Synopsis: Release the write lock.

Declaration: `procedure Endwrite`

Visibility: `public`

Description: `EndWrite` releases the current threads lock on the object, allowing other threads to acquire a read or write lock on the object.

See also: `TSimpleRWSync.BeginWrite` ([1948](#)), `TSimpleRWSync.EndRead` ([1949](#))

76.111.8 TSimpleRWSync.Beginread

Synopsis: Acquire a read lock.

Declaration: `procedure Beginread`

Visibility: `public`

Description: `BeginRead` waits till all other threads have released their read or write locks on the object, and then acquires a read lock on the object.

See also: `TSimpleRWSync.BeginWrite` ([1948](#)), `TSimpleRWSync.EndRead` ([1949](#))

76.111.9 TSimpleRWSync.Endread

Synopsis: Release the read lock.

Declaration: `procedure Endread`

Visibility: `public`

Description: `EndRead` releases the current threads read lock on the object, allowing other threads to acquire a read or write lock on the object.

See also: `TSimpleRWSync.BeginRead` ([1949](#)), `TSimpleRWSync.EndWrite` ([1948](#))

76.112 TSingleHelper

76.112.1 Description

`TSingleHelper` is the helper type for the single-sized floating point type. It contains some conversion methods, as well as access to the low-level structure of the floating-point representation of a single.

See also: `TDoubleHelper` ([1865](#)), `TExtendedHelper` ([1878](#))

76.112.2 Method overview

Page	Method	Description
1952	<code>BuildUp</code>	Build a single-sized floating point from its composing parts.
1953	<code>Exponent</code>	Exponent of the floating-point value.
1953	<code>Fraction</code>	Fraction of the floating-point value.
1950	<code>IsInfinity</code>	Check whether a value is positive or negative infinity.
1950	<code>IsNan</code>	Check whether a value equals NaN.
1950	<code>IsNegativeInfinity</code>	Check whether a value is negative infinity.
1951	<code>IsPositiveInfinity</code>	Check whether a value is positive infinity.
1953	<code>Mantissa</code>	Mantissa of the floating-point.
1951	<code>Parse</code>	Convert a string to a floating point value.
1951	<code>Size</code>	Size (in bytes) of a single-sized floating point value.
1953	<code>SpecialType</code>	Return the type of the single-sized floating point value.
1951	<code>ToString</code>	Convert a single-sized floating point value to a string.
1952	<code>TryParse</code>	Try to convert a string to a single-sized floating point value.

76.112.3 Property overview

Page	Properties	Access	Description
1954	Bytes	rw	Indexed access to the individual bytes of the floating point value.
1954	Exp	rw	The bit pattern of the exponent as stored in memory.
1955	Frac	rw	Bitpattern that makes up the fractional part.
1954	Sign	rw	Sign of the floating point value.
1954	Words	rw	Indexed access to the words that make up the floating point value.

76.112.4 TSingleHelper.IsNan

Synopsis: Check whether a value equals NaN.

Declaration: `class function IsNan(const AValue: Single) : Boolean; Overload; Static
function IsNan : Boolean; Overload`

Visibility: public

Description: `IsNan` checks whether a single-sized floating point value is NaN (Not a Number). If so, it returns `True`. When the class function version is used, the value can be specified using `AValue`. In the method version, the used value is `(Self)`.

See also: `TSingleHelper.IsInfinity` ([1950](#)), `TSingleHelper.IsPositiveInfinity` ([1951](#)), `TSingleHelper.IsNegativeInfinity` ([1950](#))

76.112.5 TSingleHelper.IsInfinity

Synopsis: Check whether a value is positive or negative infinity.

Declaration: `class function IsInfinity(const AValue: Single) : Boolean; Overload
; Static
function IsInfinity : Boolean; Overload`

Visibility: public

Description: `IsInfinity` checks whether a single-sized floating point value represents a positive or negative infinity. If so, it returns `True`. When the class function version is used, the value can be specified using `AValue`. In the method version, the used value is `(Self)`.

See also: `TSingleHelper.IsNan` ([1950](#)), `TSingleHelper.IsPositiveInfinity` ([1951](#)), `TSingleHelper.IsNegativeInfinity` ([1950](#))

76.112.6 TSingleHelper.IsNegativeInfinity

Synopsis: Check whether a value is negative infinity.

Declaration: `class function IsNegativeInfinity(const AValue: Single) : Boolean
; Overload; Static
function IsNegativeInfinity : Boolean; Overload`

Visibility: public

Description: `IsNegativeInfinity` checks whether a single-sized floating point value represents a negative infinity. If so, it returns `True`. When the class function version is used, the value can be specified using `AValue`. In the method version, the used value is `(Self)`.

See also: `TSingleHelper.IsNan` ([1950](#)), `TSingleHelper.IsPositiveInfinity` ([1951](#)), `TSingleHelper.IsInfinity` ([1950](#))

76.112.7 TSingleHelper.IsPositiveInfinity

Synopsis: Check whether a value is positive infinity.

Declaration: `class function IsPositiveInfinity(const AValue: Single) : Boolean
; Overload; Static
function IsPositiveInfinity : Boolean; Overload`

Visibility: public

Description: `IsPositiveInfinity` checks whether a single-sized floating point value represents a positive infinity. If so, it returns `True`. When the class function version is used, the value can be specified using `AValue`. In the method version, the used value is `(Self)`.

See also: `TSingleHelper.IsNan` ([1950](#)), `TSingleHelper.IsNegativeInfinity` ([1950](#)), `TSingleHelper.IsInfinity` ([1950](#))

76.112.8 TSingleHelper.Parse

Synopsis: Convert a string to a floating point value.

Declaration: `class function Parse(const AString: string) : Single; Overload; Static
class function Parse(const AString: string;
const AFormatSettings: TFormatSettings) : Single
; Overload; Static`

Visibility: public

Description: `Parse` will try to convert `AString` to a single-sized floating point value. It will take into account internationalization settings. (it uses `FloatToStr`).

Errors: If the string `AString` is not a valid floating-point value, a `EConvertError` ([1812](#)) exception is raised.

See also: `FloatToStr` ([1709](#)), `TSingleHelper.ToString` ([1951](#)), `TSingleHelper.TryParse` ([1952](#))

76.112.9 TSingleHelper.Size

Synopsis: Size (in bytes) of a single-sized floating point value.

Declaration: `class function Size : Integer; Static`

Visibility: public

Description: `Size` is the size (in bytes) of a single-sized floating point value. It is equivalent to calling `SizeOf(Single)`.

See also: `SizeOf` ([1546](#))

76.112.10 TSingleHelper.ToString

Synopsis: Convert a single-sized floating point value to a string.

Declaration: `class function ToString(const AValue: Single) : string; Overload
; Static
class function ToString(const AValue: Single;
const AFormatSettings: TFormatSettings) : string
; Overload; Static
class function ToString(const AValue: Single;
const AFormat: TFloatFormat;
const APrecision: Integer;`


```

        const ADigits: Integer) : string; Overload
        ; Static
class function ToString(const AValue: Single;
        const AFormat: TFloatFormat;
        const APrecision: Integer;
        const ADigits: Integer;
        const AFormatSettings: TFormatSettings) : string
        ; Overload; Static
function ToString(const AFormat: TFloatFormat;
        const APrecision: Integer; const ADigits: Integer)
        : string; Overload
function ToString(const AFormat: TFloatFormat;
        const APrecision: Integer; const ADigits: Integer;
        const AFormatSettings: TFormatSettings) : string
        ; Overload
function ToString(const AFormatSettings: TFormatSettings) : string
        ; Overload
function ToString : string; Overload

```

Visibility: public

Description: ToString will convert AValue (or Self in the plain method version) to a string. Optionally FormatSettings can be specified, to be able to specify the decimal separator character to use.

Additionally, a precision APrecision and number of digits ADigits can be specified, in conjunction with a AFormat parameter to specify the form in which the floating-point value must be represented. (see TFloatFormat (1632) for an explanation of the various values). In this case, FloatToStrF (1710) is used to format the value. In the absence of these parameters, FloatToStr (1709) is called.

See also: FloatToStr (1709), FloatToStrF (1710), TFloatFormat (1632)

76.112.11 TSingleHelper.TryParse

Synopsis: Try to convert a string to a single-sized floating point value.

Declaration:

```

class function TryParse(const AString: string; out AValue: Single)
        : Boolean; Overload; Static
class function TryParse(const AString: string; out AValue: Single;
        const AFormatSettings: TFormatSettings) : Boolean
        ; Overload; Static

```

Visibility: public

Description: TryParse attempts to convert the string AString to a single-sized floating point value and reports True if the conversion was successful. In that case the parsed value is returned in AValue.

If the conversion failed, False is returned.

See also: TSingleHelper.Parse (1951), TSingleHelper.ToString (1951)

76.112.12 TSingleHelper.BuildUp

Synopsis: Build a single-sized floating point from its composing parts.

Declaration:

```

procedure BuildUp(const ASignFlag: Boolean; const AMantissa: QWord;
        const AExponent: Integer)

```

Visibility: public

Description: `BuildUp` will compose a single-sized floating point value from the sign `ASignFlag`, mantissa `AMantissa` and exponent `AExponent`. It simply sets the `Sign` (1954), `Exp` (1954) and `Frac` (1955) properties in 1 call.

See also: `TSingleHelper.Sign` (1954), `TSingleHelper.Exp` (1954), `TSingleHelper.Frac` (1955)

76.112.13 TSingleHelper.Exponent

Synopsis: Exponent of the floating-point value.

Declaration: `function Exponent : Integer`

Visibility: public

Description: `Exponent` is the value X in the representation of the floating-point value in $m \cdot 2^{\hat{X}}$, i.e. the exponent.

See also: `TSingleHelper.Sign` (1954), `TSingleHelper.Exp` (1954), `TSingleHelper.Frac` (1955), `TSingleHelper.Fraction` (1953), `TSingleHelper.Mantissa` (1953)

76.112.14 TSingleHelper.Fraction

Synopsis: Fraction of the floating-point value.

Declaration: `function Fraction : Extended`

Visibility: public

Description: `Fraction` is the decimal part of the floating-point value.

See also: `TSingleHelper.Sign` (1954), `TSingleHelper.Exp` (1954), `TSingleHelper.Exponent` (1953), `TSingleHelper.Frac` (1955), `TSingleHelper.Mantissa` (1953)

76.112.15 TSingleHelper.Mantissa

Synopsis: Mantissa of the floating-point.

Declaration: `function Mantissa : QWord`

Visibility: public

Description: `Mantissa` is the value of the significant without the hidden bit. This means it the plain bit pattern as it is stored in memory.

See also: `TSingleHelper.Sign` (1954), `TSingleHelper.Exp` (1954), `TSingleHelper.Exponent` (1953), `TSingleHelper.Frac` (1955), `TSingleHelper.Fraction` (1953)

76.112.16 TSingleHelper.SpecialType

Synopsis: Return the type of the single-sized floating point value.

Declaration: `function SpecialType : TFloatSpecial`

Visibility: public

Description: `SpecialType` checks whether the single-sized floating point value equals one of several special values, and returns an enumerated value describing which value this is. See `TFloatSpecial` (1609) for an explanation of the possible values.

See also: `TFloatSpecial` (1609)

76.112.17 TSingleHelper.Bytes

Synopsis: Indexed access to the individual bytes of the floating point value.

Declaration: `Property Bytes[AIndex: Cardinal]: Byte`

Visibility: public

Access: Read,Write

Description: `Bytes` can be used to get or set the various bytes that make up the single-sized floating point value.
The index runs from 0 to `Size-1`.

See also: `TSingleHelper.Words` ([1954](#)), `TSingleHelper.Size` ([1951](#))

76.112.18 TSingleHelper.Words

Synopsis: Indexed access to the words that make up the floating point value.

Declaration: `Property Words[AIndex: Cardinal]: Word`

Visibility: public

Access: Read,Write

Description: `Words` can be used to get or set the various bytes that make up the single-sized floating point value.
The index runs from 0 to $(\text{Size}-1) \div 2$.

See also: `TSingleHelper.Bytes` ([1954](#)), `TSingleHelper.Size` ([1951](#))

76.112.19 TSingleHelper.Sign

Synopsis: Sign of the floating point value.

Declaration: `Property Sign : Boolean`

Visibility: public

Access: Read,Write

Description: `Sign` returns `True` if the sign bit of the value is set (i.e. it is a negative value) or `False` if it is not set (i.e. it is a positive value).

See also: `TSingleHelper.Bytes` ([1954](#)), `TSingleHelper.Exp` ([1954](#)), `TSingleHelper.Frac` ([1955](#)), `TSingleHelper.Mantissa` ([1953](#)), `TSingleHelper.Fraction` ([1953](#)), `TSingleHelper.Exponent` ([1953](#))

76.112.20 TSingleHelper.Exp

Synopsis: The bit pattern of the exponent as stored in memory.

Declaration: `Property Exp : QWord`

Visibility: public

Access: Read,Write

Description: `Exp` is the internal representation of the Exponent ([1953](#)).

See also: `TSingleHelper.Bytes` ([1954](#)), `TSingleHelper.Sign` ([1954](#)), `TSingleHelper.Frac` ([1955](#)), `TSingleHelper.Mantissa` ([1953](#)), `TSingleHelper.Fraction` ([1953](#)), `TSingleHelper.Exponent` ([1953](#))

76.112.21 TSingleHelper.Frac

Synopsis: Bitpattern that makes up the fractional part.

Declaration: `Property Frac : QWord`

Visibility: public

Access: Read,Write

Description: `Frac` is the bit pattern representing the fractional part (significand) including the preceding 1 (the hidden bit).

See also: `TSingleHelper.Bytes` (1954), `TSingleHelper.Sign` (1954), `TSingleHelper.Exp` (1954), `TSingleHelper.Mantissa` (1953), `TSingleHelper.Fraction` (1953), `TSingleHelper.Exponent` (1953)

76.113 TSmallIntHelper**76.113.1 Description**

`TSmallIntHelper` contains some auxiliary routines for a `SmallInt`-typed ordinal value. It consists mainly of conversion routines to and from other types.

See also: `TStringHelper` (1609), `TShortIntHelper` (1928), `TByteHelper` (1851), `TWordHelper` (2000), `TCardinalHelper` (1857), `TIntegerHelper` (1893), `TInt64Helper` (1886), `TQWordHelper` (1921), `TNativeIntHelper` (1908), `TNativeUIntHelper` (1915)

76.113.2 Method overview

Page	Method	Description
1959	<code>Clear</code>	Set value to 0.
1958	<code>ClearBit</code>	Set bit to 0.
1959	<code>HighestSetBitPos</code>	Return the position of the leftmost bit set.
1960	<code>LowestSetBitPos</code>	Return the position of the rightmost bit set.
1956	<code>Parse</code>	Convert from a string.
1958	<code>SetBit</code>	Set bit to 1.
1960	<code>SetBitsCount</code>	Count number of bits set.
1956	<code>Size</code>	Size, in bytes, of the <code>SmallInt</code> value.
1959	<code>TestBit</code>	Check bit.
1957	<code>ToBinString</code>	Convert to a binary string representation.
1957	<code>ToBoolean</code>	Convert to a boolean value.
1958	<code>ToDouble</code>	Convert to a double-sized floating point value.
1958	<code>ToExtended</code>	Convert to an extended-sized floating point value.
1959	<code>ToggleBit</code>	Invert bit.
1957	<code>ToHexString</code>	Convert to a hexadecimal string representation.
1957	<code>ToSingle</code>	Convert to a single-sized floating point value.
1956	<code>ToString</code>	Convert the value to string.
1956	<code>TryParse</code>	Try to convert a string to a <code>SmallInt</code> , report success or failure.

76.113.3 Property overview

Page	Properties	Access	Description
1960	<code>Bits</code>	rw	Read or write a bit.
1961	<code>Bytes</code>	rw	Read or write a byte.
1961	<code>Nibbles</code>	rw	Read or write a nibble.

76.113.4 TSmallIntHelper.Parse

Synopsis: Convert from a string.

Declaration: `class function Parse(const AString: string) : SmallInt; Static`

Visibility: public

Description: Parse will attempt to convert the string `AString` to a `SmallInt` value. It uses the `StrToInt` ([1773](#)) function to perform the conversion, so no localization is taken into account.

Errors: If the string does not contain a valid `SmallInt` value, an `EConvertError` (1812) exception is raised.

See also: [TSmallIntHelper.ToString \(1956\)](#), [TSmallIntHelper.TryParse \(1956\)](#), [StrToInt \(1773\)](#)

76.113.5 TSmallIntHelper.Size

Synopsis: Size, in bytes, of the SmallInt value.

```
Declaration: class function Size : Integer; Static
```

Visibility: public

Description: `Size` returns the size (in `SmallInts`) of the `SmallInt` value. This is equivalent to `SizeOf (SmallInt)`.

Errors: None.

See also: [SizeOf \(1546\)](#)

76.113.6 TSmallIntHelper.ToString

Synopsis: Convert the value to string.

```
Declaration: class function ToString(const AValue: SmallInt) : string; Overload
              ; Static
              function ToString : string; Overload
```

Visibility: public

Description: ToString will, in the class function variant of this method, convert AValue to a string representation. In the regular method overloaded version of ToString, the SmallInt value itself is used. The IntToStr (1738) function is used to do the conversion.

See also: [TSmallIntHelper.Parse \(1956\)](#), [IntToStr \(1738\)](#)

76.113.7 TSmallIntHelper.TryParse

Synopsis: Try to convert a string to a SmallInt, report success or failure.

```
Declaration: class function TryParse(const AString: string; out AValue: SmallInt)
                                     : Boolean; Static
```

Visibility: public

Description: TryParse attempts to convert the string `AString` to a `SmallInt`, and reports the success of the attempt. If the attempt is successful, then `True` is returned, and the actual value of the `SmallInt` is returned in `AValue`.

It uses the `val (1609)` function to perform the conversion, so no localization is taken into account.

See also: [TSmallIntHelper.Parse \(1956\)](#), [Val \(1570\)](#)

76.113.8 TSmallIntHelper.ToBoolean

Synopsis: Convert to a boolean value.

Declaration: `function ToBoolean : Boolean`

Visibility: public

Description: `ToBoolean` converts the `SmallInt` value to a boolean: it returns `True` if the value is nonzero, `False` if it is zero.

See also: `TSmallIntHelper.ToSingle` (1957), `TSmallIntHelper.ToDouble` (1958), `TSmallIntHelper.ToExtended` (1958), `TSmallIntHelper.ToString` (1956), `TSmallIntHelper.ToHexString` (1957)

76.113.9 TSmallIntHelper.ToBinString

Synopsis: Convert to a binary string representation.

Declaration: `function ToBinString : string`

Visibility: public

Description: `ToBinString` converts the `smallint` value to a binary string representation, showing all 16 bits.

See also: `TSmallIntHelper.ToHexString` (1957), `TSmallIntHelper.ToString` (1956)

76.113.10 TSmallIntHelper.ToHexString

Synopsis: Convert to a hexadecimal string representation.

Declaration: `function ToHexString : string; Overload`
`function ToHexString(const AMinDigits: Integer) : string; Overload`

Visibility: public

Description: `ToHexString` converts the `SmallInt` value to a hexadecimal string representation. The `AMinDigits` argument specifies the minimal number of characters in the resulting string. The string will be left-padded with zeroes if the representation contains less than `AMinDigits` characters.

See also: `TSmallIntHelper.ToBoolean` (1957), `TSmallIntHelper.ToSingle` (1957), `TSmallIntHelper.ToDouble` (1958), `TSmallIntHelper.ToString` (1956), `TSmallIntHelper.ToExtended` (1958)

76.113.11 TSmallIntHelper.ToSingle

Synopsis: Convert to an single-sized floating point value.

Declaration: `function ToSingle : Single`

Visibility: public

Description: `ToSingle` converts the `SmallInt` value to a single-sized floating point value.

See also: `TSmallIntHelper.ToBoolean` (1957), `TSmallIntHelper.ToDouble` (1958), `TSmallIntHelper.ToExtended` (1958), `TSmallIntHelper.ToString` (1956), `TSmallIntHelper.ToHexString` (1957)

76.113.12 TSmallIntHelper.ToDouble

Synopsis: Convert to a double-sized floating point value.

Declaration: `function ToDouble : Double`

Visibility: public

Description: `ToDouble` converts the `SmallInt` value to a double-sized floating point value

See also: `TSmallIntHelper.ToBoolean` ([1957](#)), `TSmallIntHelper.ToExtended` ([1958](#)), `TSmallIntHelper.ToSingle` ([1957](#)), `TSmallIntHelper.ToString` ([1956](#)), `TSmallIntHelper.ToHexString` ([1957](#))

76.113.13 TSmallIntHelper.ToExtended

Synopsis: Convert to an extended-sized floating point value.

Declaration: `function ToExtended : Extended`

Visibility: public

Description: `ToDouble` converts the `SmallInt` value to an extended-sized floating point value.

See also: `TSmallIntHelper.ToBoolean` ([1957](#)), `TSmallIntHelper.ToSingle` ([1957](#)), `TSmallIntHelper.ToDouble` ([1958](#)), `TSmallIntHelper.ToString` ([1956](#)), `TSmallIntHelper.ToHexString` ([1957](#))

76.113.14 TSmallIntHelper.SetBit

Synopsis: Set bit to 1.

Declaration: `function SetBit(const Index: TSmallIntBitIndex) : SmallInt`

Visibility: public

Description: `SetBit` puts value 1 to bit number determined by argument `Index`.

See also: `TSmallIntHelper.ClearBit` ([1958](#)), `TSmallIntHelper.ToggleBit` ([1959](#)), `TSmallIntHelper.TestBit` ([1959](#)), `TSmallIntHelper.HighestSetBitPos` ([1959](#)), `TSmallIntHelper.LowestSetBitPos` ([1960](#)), `TSmallIntHelper.SetBitsCount` ([1960](#)), `TSmallIntHelper.Bits` ([1960](#))

76.113.15 TSmallIntHelper.ClearBit

Synopsis: Set bit to 0.

Declaration: `function ClearBit(const Index: TSmallIntBitIndex) : SmallInt`

Visibility: public

Description: `ClearBit` puts value 0 to bit number determined by argument `Index`.

See also: `TSmallIntHelper.SetBit` ([1958](#)), `TSmallIntHelper.ToggleBit` ([1959](#)), `TSmallIntHelper.TestBit` ([1959](#)), `TSmallIntHelper.HighestSetBitPos` ([1959](#)), `TSmallIntHelper.LowestSetBitPos` ([1960](#)), `TSmallIntHelper.SetBitsCount` ([1960](#)), `TSmallIntHelper.Bits` ([1960](#))

76.113.16 TSmallIntHelper.ToggleBit

Synopsis: Invert bit.

Declaration: `function ToggleBit (const Index: TSmallIntBitIndex) : SmallInt`

Visibility: public

Description: `ToggleBit` reads bit value from bit number determined by argument `Index`, inverts it (if it was 0 it becomes 1, and vice versa), and stores it back.

See also: `TSmallIntHelper.SetBit` (1958), `TSmallIntHelper.ToggleBit` (1959), `TSmallIntHelper.TestBit` (1959), `TSmallIntHelper.HighestSetBitPos` (1959), `TSmallIntHelper.LowestSetBitPos` (1960), `TSmallIntHelper.SetBitsCount` (1960), `TSmallIntHelper.Bits` (1960)

76.113.17 TSmallIntHelper.TestBit

Synopsis: Check bit.

Declaration: `function TestBit (const Index: TSmallIntBitIndex) : Boolean`

Visibility: public

Description: `TestBit` reads boolean value (true if bit is 1, and false if bit is 0) from bit number determined by argument `Index`.

See also: `TSmallIntHelper.SetBit` (1958), `TSmallIntHelper.ToggleBit` (1959), `TSmallIntHelper.TestBit` (1959), `TSmallIntHelper.HighestSetBitPos` (1959), `TSmallIntHelper.LowestSetBitPos` (1960), `TSmallIntHelper.SetBitsCount` (1960), `TSmallIntHelper.Bits` (1960)

76.113.18 TSmallIntHelper.Clear

Synopsis: Set value to 0.

Declaration: `procedure Clear`

Visibility: public

76.113.19 TSmallIntHelper.HighestSetBitPos

Synopsis: Return the position of the leftmost bit set.

Declaration: `function HighestSetBitPos : Int8`

Visibility: public

Description: `HighestSetBitPos` scans the `smallint`, starting at the leftmost position and working towards position 0, and returns the index of the first set bit. The position is measured from the 0-th, rightmost bit. When there are no bits set, the result is -1.

See also: `System.BsrByte` (1609), `TSmallIntHelper.SetBit` (1958), `TSmallIntHelper.ClearBit` (1958), `TSmallIntHelper.ToggleBit` (1959), `TSmallIntHelper.TestBit` (1959), `TSmallIntHelper.LowestSetBitPos` (1960), `TSmallIntHelper.SetBitsCount` (1960), `TSmallIntHelper.Bits` (1960)

76.113.20 TSmallIntHelper.LowestSetBitPos

Synopsis: Return the position of the rightmost bit set.

Declaration: `function LowestSetBitPos : Int8`

Visibility: public

Description: `LowestSetBitPos` scans the `smallint`, starting at position 0 (rightmost position) and returns the index of the first set bit. The position is measured from the 0-th, rightmost bit. When there are no bits set, the result is -1.

See also: `System.BsfByte` (1609), `TSmallIntHelper.SetBit` (1958), `TSmallIntHelper.ClearBit` (1958), `TSmallIntHelper.ToggleBit` (1959), `TSmallIntHelper.TestBit` (1959), `TSmallIntHelper.HighestSetBitPos` (1959), `TSmallIntHelper.SetBitsCount` (1960), `TSmallIntHelper.Bits` (1960)

76.113.21 TSmallIntHelper.SetBitsCount

Synopsis: Count number of bits set.

Declaration: `function SetBitsCount : Byte`

Visibility: public

Description: `SetBitsCount` returns the number of bits set in a `smallint`.

See also: `System.PopCnt` (1609), `TSmallIntHelper.SetBit` (1958), `TSmallIntHelper.ClearBit` (1958), `TSmallIntHelper.ToggleBit` (1959), `TSmallIntHelper.TestBit` (1959), `TSmallIntHelper.HighestSetBitPos` (1959), `TSmallIntHelper.LowestSetBitPos` (1960), `TSmallIntHelper.Bits` (1960)

76.113.22 TSmallIntHelper.Bits

Synopsis: Read or write a bit.

Declaration: `Property Bits[aIndex: TSmallIntBitIndex]: Boolean`

Visibility: public

Access: Read,Write

Description: `Bits` property reads or writes boolean value to a bit number determined by argument `aIndex`.

```
MySmallInt.Clear; // %000000000000000000 MySmallInt equals 0
MySmallInt.Bits[0] := true; // %000000000000000001 MySmallInt equals 1
MySmallInt.Bits[2] := true; // %00000000000000101 MySmallInt equals 5
WriteLn(MySmallInt.Bits[2]); // TRUE
```

See also: `TSmallIntHelper.SetBit` (1958), `TSmallIntHelper.ClearBit` (1958), `TSmallIntHelper.ToggleBit` (1959), `TSmallIntHelper.TestBit` (1959), `TSmallIntHelper.HighestSetBitPos` (1959), `TSmallIntHelper.LowestSetBitPos` (1960), `TSmallIntHelper.SetBitsCount` (1960), `TSmallIntHelper.Nibbles` (1961), `TSmallIntHelper.Bytes` (1961)

76.113.23 TSmallIntHelper.Nibbles

Synopsis: Read or write a nibble.

Declaration: `Property Nibbles[aIndex: TSmallIntNibbleIndex]: nibble`

Visibility: public

Access: Read,Write

Description: Nibbles property reads or writes nibble value to a nibble number determined by argument aIndex.

```
MySmallInt.Clear; // %0000000000000000 MySmallInt equals 0
MySmallInt.Nibbles[1] := 3; // %000000000000110000 MySmallInt equals 48
WriteLn(MySmallInt.Nibbles[1]); // 3
```

See also: TSmallIntHelper.Bits ([1960](#)), TSmallIntHelper.Bytes ([1961](#))

76.113.24 TSmallIntHelper.Bytes

Synopsis: Read or write a byte.

Declaration: `Property Bytes[aIndex: TSmallIntByteIndex]: Byte`

Visibility: public

Access: Read,Write

Description: Bytes property reads or writes byte value to a byte number determined by argument aIndex.

```
MySmallInt.Clear; // %0000000000000000 MySmallInt equals 0
MySmallInt.Bytes[1] := 1; // %00000000100000000 MySmallInt equals 256
WriteLn(MySmallInt.Bytes[1]); // 1
```

See also: TSmallIntHelper.Bits ([1960](#)), TSmallIntHelper.Nibbles ([1961](#))

76.114 TUnicodeEncoding

76.114.1 Description

TUnicodeEncoding is the encoding class used to represent the UTF-16 encoding.

See also: TUnicodeEncoding ([1961](#)), TUTF7Encoding ([1982](#)), TMBCSEncoding ([1904](#)), TBigendianUnicodeEncoding ([1846](#))

76.114.2 Method overview

Page	Method	Description
1962	Clone	Clone a TUnicodeEncoding instance.
1962	Create	Create a new instance of the TUnicodeEncoding class.
1962	GetMaxByteCount	Return max number of bytes needed to represent a string.
1962	GetMaxCharCount	Return max number of characters that can be represented by an array of bytes.
1963	GetPreamble	Return BOM marker bytes.

76.114.3 TUnicodeEncoding.Create

Synopsis: Create a new instance of the TUnicodeEncoding class.

Declaration: constructor Create; Virtual

Visibility: public

Description: Create creates a new instance of the TUnicodeEncoding class and sets the codepage to CP_UTF16.

See also: TEncoding.CodePage ([1875](#)), TEncoding ([1871](#))

76.114.4 TUnicodeEncoding.Clone

Synopsis: Clone a TUnicodeEncoding instance.

Declaration: function Clone : TEncoding; Override

Visibility: public

Description: Clone overrides TEncoding.Clone ([1872](#)) to provide a clone of the TUnicodeEncoding instance.

See also: TEncoding.Clone ([1872](#))

76.114.5 TUnicodeEncoding.GetMaxByteCount

Synopsis: Return max number of bytes needed to represent a string.

Declaration: function GetMaxByteCount(CharCount: Integer) : Integer; Override

Visibility: public

Description: GetMaxByteCount overrides TEncoding.GetMaxByteCount ([1874](#)) to return the maximum number of bytes needed to represent a string.

See also: TEncoding.GetMaxByteCount ([1874](#))

76.114.6 TUnicodeEncoding.GetMaxCharCount

Synopsis: Return max number of characters that can be represented by an array of bytes.

Declaration: function GetMaxCharCount(ByteCount: Integer) : Integer; Override

Visibility: public

Description: GetMaxCharCount overrides TEncoding.GetMaxCharCount ([1874](#)) to return the maximum number of bytes needed to represent a string.

See also: TEncoding.GetMaxCharCount ([1874](#))

76.114.7 TUnicodeEncoding.GetPreamble

Synopsis: Return BOM marker bytes.

Declaration: `function GetPreamble : TBytes; Override`

Visibility: `public`

Description: `GetPreamble` overrides `TEncoding.GetPreamble` (1874) to return the 2 UTF-16 BOM Marker bytes (\$FF,\$FE).

See also: `TEncoding.GetPreamble` (1874)

76.115 TUNICODESTRINGBUILDER

76.115.1 Description

`TUnicodeStringBuilder` is a class to construct an anstring out of other strings or characters. It's methods can be used instead of constructing the string manually using the normal `+` or `concat` operators. If used properly (e.g. by setting a sufficiently large `Capacity` (1969)), will result in faster operation. It offers various methods to append to the string.

To create a unicode string, use the `TUnicodeStringBuilder` (1963) class instead.

See also: `TUnicodeStringBuilder` (1963)

76.115.2 Method overview

Page	Method	Description
1964	<code>Append</code>	Append data in string form to the buffer.
1965	<code>AppendFormat</code>	Append the result of a <code>Format</code> .
1965	<code>AppendLine</code>	Append a string followed by a newline.
1965	<code>Clear</code>	Clear the string being built.
1966	<code>CopyTo</code>	Copy string bytes to an array.
1963	<code>Create</code>	Create a new <code>TUnicodeStringBuilder</code> instance.
1966	<code>EnsureCapacity</code>	Set the capacity if needed.
1966	<code>Equals</code>	Check if 2 stringbuilders have the same content string.
1966	<code>GetEnumerator</code>	
1967	<code>Insert</code>	Insert data in string form into the buffer at a given position.
1968	<code>Remove</code>	Remove data from the string.
1968	<code>Replace</code>	Replace a range of characters.
1968	<code>ToString</code>	Return the string buffer as an <code>UnicodeString</code> .

76.115.3 Property overview

Page	Properties	Access	Description
1969	<code>Capacity</code>	<code>rw</code>	Current capacity of the string buffer.
1969	<code>Chars</code>	<code>rw</code>	Indexed access to the characters in the string.
1969	<code>Length</code>	<code>rw</code>	Current length of the string buffer.
1969	<code>MaxCapacity</code>	<code>r</code>	Maximum capacity of the string buffer.

76.115.4 TUNICODESTRINGBUILDER.Create

Synopsis: Create a new `TUnicodeStringBuilder` instance.

Declaration: constructor Create

```

constructor Create(aCapacity: Integer)
constructor Create(const AValue: UnicodeString)
constructor Create(aCapacity: Integer; aMaxCapacity: Integer)
constructor Create(const AValue: UnicodeString; aCapacity: Integer)
constructor Create(const AValue: UnicodeString; StartIndex: Integer;
                  aLength: Integer; aCapacity: Integer)

```

Visibility: public

Description: Create creates an empty TUnicodeStringBuilder instance. It can optionally set the initial capacity to aCapacity and the maximum capacity to aMaxCapacity and can also initialize the string buffer with characters from aValue: aLength characters starting at StartIndex (1-based) will be copied to the string buffer. If aLength and StartIndex are omitted, then all characters are copied.

See also: Capacity ([1969](#)), MaxCapacity ([1969](#))

76.115.5 TUNICODESTRINGBUILDER.Append

Synopsis: Append data in string form to the buffer.

Declaration: function Append(const AValue: Boolean) : TUNICODESTRINGBUILDER

```

function Append(const AValue: Byte) : TUNICODESTRINGBUILDER
function Append(const AValue: WideChar) : TUNICODESTRINGBUILDER
function Append(const AValue: Currency) : TUNICODESTRINGBUILDER
function Append(const AValue: Double) : TUNICODESTRINGBUILDER
function Append(const AValue: SmallInt) : TUNICODESTRINGBUILDER
function Append(const AValue: LongInt) : TUNICODESTRINGBUILDER
function Append(const AValue: Int64) : TUNICODESTRINGBUILDER
function Append(const AValue: TObject) : TUNICODESTRINGBUILDER
function Append(const AValue: ShortInt) : TUNICODESTRINGBUILDER
function Append(const AValue: Single) : TUNICODESTRINGBUILDER
function Append(const AValue: UInt64) : TUNICODESTRINGBUILDER
function Append(const AValue: Array of WideChar) : TUNICODESTRINGBUILDER
function Append(const AValue: Word) : TUNICODESTRINGBUILDER
function Append(const AValue: Cardinal) : TUNICODESTRINGBUILDER
function Append(const AValue: PWideChar) : TUNICODESTRINGBUILDER
function Append(const AValue: UnicodeString) : TUNICODESTRINGBUILDER
function Append(const AValue: RawByteString) : TUNICODESTRINGBUILDER
function Append(const AValue: WideChar; RepeatCount: Integer)
    : TUNICODESTRINGBUILDER
function Append(const AValue: Array of WideChar; StartIndex: Integer;
                SBCharCount: Integer) : TUNICODESTRINGBUILDER
function Append(const AValue: UnicodeString; StartIndex: Integer;
                Count: Integer) : TUNICODESTRINGBUILDER
function Append(const Fmt: UnicodeString; const Args: Array of const)
    : TUNICODESTRINGBUILDER

```

Visibility: public

Description: Append in its various overloaded forms will append the aValue argument to the buffer, after converting it to a string value if necessary:

- For integer values the various IntToStr ([1738](#)) functions will be used.

- For `TObject` (1609) the `TObject.ToString` (1609) is used.
- The `repeatCount` argument will repeat the character as many times as instructed using `StringOfChar` (1609).
- If the `StartIndex` and `SBCharCount` are specified, then `SBCharCount` characters are copied starting from position `StartIndex`.

An overloaded version exists with the same arguments as `AppendFormat` (1965).

See also: `StringOfChar` (1609), `AppendFormat` (1965), `AppendLine` (1965), `Insert` (1967), `Remove` (1968)

76.115.6 TUNICODESTRINGBUILDER.AppendFormat

Synopsis: Append the result of a `Format`.

Declaration: `function AppendFormat(const Fmt: UnicodeString;
const Args: Array of const) : TUNICODESTRINGBUILDER`

Visibility: public

Description: `AppendFormat` calls `Format` (1715) with `fmt` and `Args` and calls `append` (1964) with the resulting string.

Errors: the call to `Format` may result in an exception if the format parameters do not match the passed arguments.

See also: `Append` (1964), `Insert` (1967), `Remove` (1968), `Format` (1715)

76.115.7 TUNICODESTRINGBUILDER.AppendLine

Synopsis: Append a string followed by a newline.

Declaration: `function AppendLine : TUNICODESTRINGBUILDER
function AppendLine(const AValue: RawByteString) : TUNICODESTRINGBUILDER`

Visibility: public

Description: `AppendLine` will append `aValue` (if it is specified) and appends the OS linebreak character (`sLineBreak` (1361)).

See also: `TUnicodeStringBuilder.AppendFormat` (1965), `TUnicodeStringBuilder.Append` (1964), `#rtl.system.sLineBreak` (1361), `Remove` (1968), `Insert` (1967)

76.115.8 TUNICODESTRINGBUILDER.Clear

Synopsis: Clear the string being built.

Declaration: `procedure Clear`

Visibility: public

Description: `Clear` sets the length of the string to 0 and sets the `Capacity` (1969) to the default capacity (64 bytes).

See also: `Capacity` (1969), `Remove` (1968)

76.115.9 TUNICODESTRINGBUILDER.CopyTo

Synopsis: Copy string bytes to an array.

Declaration: `procedure CopyTo(SourceIndex: Integer;
var Destination: Array of WideChar;
DestinationIndex: Integer; Count: Integer)`

Visibility: public

Description: `CopyTo` copies `aCount` characters from the string, starting at position `SourceIndex` (0-based) in the string to build, to the character array `Destination` at position `DestinationIndex`.

Errors: If the range of characters to copy falls outside the allowed range of characters in either source (available characters) or destination (available room for characters), then an `ERangeError` (1820) exception is raised.

See also: `ERangeError` (1820)

76.115.10 TUNICODESTRINGBUILDER.EnsureCapacity

Synopsis: Set the capacity if needed.

Declaration: `function EnsureCapacity(aCapacity: Integer) : Integer`

Visibility: public

Description: `EnsureCapacity` checks that `aCapacity` is in the allowed range (less than `MaxCapacity` (1969)) and if the current capacity is less than `aCapacity`, increases the capacity to `aCapacity`.

Errors: If `aCapacity` is larger than `MaxCapacity` (1969) or negative, an `ERangeError` (1820) exception is raised.

See also: `MaxCapacity` (1969)

76.115.11 TUNICODESTRINGBUILDER.Equals

Synopsis: Check if 2 stringbuilders have the same content string.

Declaration: `function Equals(StringBuilder: TUNICODESTRINGBUILDER) : Boolean
; Reintroduce`

Visibility: public

Description: `Equals` checks that the self instance has the same content as the `StringBuilder` instance. It checks the length and content (as bytes) of the strings.

See also: `TUnicodeStringBuilder.Clear` (1965)

76.115.12 TUNICODESTRINGBUILDER.GetEnumerator

Declaration: `function GetEnumerator : TCharEnumerator`

Visibility: public

76.115.13 TUNICODESTRINGBUILDER.Insert

Synopsis: Insert data in string form into the buffer at a given position.

Declaration: `function Insert(Index: Integer; const AValue: Boolean)`
 `: TUNICODESTRINGBUILDER`
`function Insert(Index: Integer; const AValue: Byte)`
 `: TUNICODESTRINGBUILDER`
`function Insert(Index: Integer; const AValue: WideChar)`
 `: TUNICODESTRINGBUILDER`
`function Insert(Index: Integer; const AValue: Currency)`
 `: TUNICODESTRINGBUILDER`
`function Insert(Index: Integer; const AValue: Double)`
 `: TUNICODESTRINGBUILDER`
`function Insert(Index: Integer; const AValue: SmallInt)`
 `: TUNICODESTRINGBUILDER`
`function Insert(Index: Integer; const AValue: LongInt)`
 `: TUNICODESTRINGBUILDER`
`function Insert(Index: Integer; const AValue: Array of WideChar)`
 `: TUNICODESTRINGBUILDER`
`function Insert(Index: Integer; const AValue: Int64)`
 `: TUNICODESTRINGBUILDER`
`function Insert(Index: Integer; const AValue: TObject)`
 `: TUNICODESTRINGBUILDER`
`function Insert(Index: Integer; const AValue: ShortInt)`
 `: TUNICODESTRINGBUILDER`
`function Insert(Index: Integer; const AValue: Single)`
 `: TUNICODESTRINGBUILDER`
`function Insert(Index: Integer; const AValue: UnicodeString)`
 `: TUNICODESTRINGBUILDER`
`function Insert(Index: Integer; const AValue: Word)`
 `: TUNICODESTRINGBUILDER`
`function Insert(Index: Integer; const AValue: Cardinal)`
 `: TUNICODESTRINGBUILDER`
`function Insert(Index: Integer; const AValue: UInt64)`
 `: TUNICODESTRINGBUILDER`
`function Insert(Index: Integer; const AValue: UnicodeString;`
 `const aRepeatCount: Integer) : TUNICODESTRINGBUILDER`
`function Insert(Index: Integer; const AValue: Array of WideChar;`
 `startIndex: Integer; SBCharCount: Integer)`
 `: TUNICODESTRINGBUILDER`

Visibility: public

Description: Insert in its various overloaded forms will insert the `aValue` argument into the buffer at position `Index` (0-based), after converting it to a string value if necessary:

- For integer values the various `IntToStr` (1738) functions will be used.
- For `TObject` (1609) the `TObject.ToString` (1609) is used.
- The `repeatCount` argument will insert the character as many times as instructed.
- If the `startIndex` and `SBCharCount` are specified, then `SBCharCount` characters are copied starting from position `startIndex`.

See also: `AppendFormat` (1965), `AppendLine` (1965), `Append` (1964), `Remove` (1968), `Replace` (1968)

76.115.14 TUNICODESTRINGBUILDER.Remove

Synopsis: Remove data from the string.

Declaration: `function Remove(StartIndex: Integer; RemLength: Integer)
: TUNICODESTRINGBUILDER`

Visibility: public

Description: Remove will remove `RemLength` characters from the string buffer, starting at position `StartIndex` (0-based).

Errors: If the range of characters to remove falls outside the allowed range of characters, then an `ERangeError` (1820) exception is raised.

See also: `AppendFormat` (1965), `AppendLine` (1965), `Append` (1964), `Insert` (1967), `Clear` (1965)

76.115.15 TUNICODESTRINGBUILDER.Replace

Synopsis: Replace a range of characters.

Declaration: `function Replace(const OldChar: WideChar; const NewChar: WideChar)
: TUNICODESTRINGBUILDER`
`function Replace(const OldChar: WideChar; const NewChar: WideChar;
StartIndex: Integer; Count: Integer)
: TUNICODESTRINGBUILDER`
`function Replace(const OldValue: UnicodeString;
const NewValue: UnicodeString) : TUNICODESTRINGBUILDER`
`function Replace(const OldValue: UnicodeString;
const NewValue: UnicodeString; StartIndex: Integer;
Count: Integer) : TUNICODESTRINGBUILDER`

Visibility: public

Description: Replace replaces occurrences of `OldValue` with the characters in `NewValue`. The search operation starts at position `StartIndex` (0-based, default 0) till position `StartIndex+aCount` is reached.

Errors: If `StartIndex` or `StartIndex+aCount` is out of range, an `ERangeError` (1820) exception is raised.

See also: `AppendFormat` (1965), `AppendLine` (1965), `Append` (1964), `Insert` (1967), `Clear` (1965), `Remove` (1968)

76.115.16 TUNICODESTRINGBUILDER.ToString

Synopsis: Return the string buffer as an `UnicodeString`.

Declaration: `function ToString : UnicodeString`
`function ToString(aStartIndex: Integer; aLength: Integer)
: UnicodeString; Reintroduce`

Visibility: public

Description: `ToString` returns the current content of the buffer as an `UnicodeString`.

See also: `Clear` (1965), `CopyTo` (1966)

76.115.17 TUNICODESTRINGBUILDER.Chars

Synopsis: Indexed access to the characters in the string.

Declaration: `Property Chars[index: Integer]: WideChar; default`

Visibility: public

Access: Read,Write

Description: `Chars` allows indexed access to the characters in the string using a zero-based `Index`. The characters can be read or written. The allowed range for `Index` is 0 to `Length` (1969)-1.

See also: `Length` (1969)

76.115.18 TUNICODESTRINGBUILDER.Length

Synopsis: Current length of the string buffer.

Declaration: `Property &Length : Integer`

Visibility: public

Access: Read,Write

Description: `Length` can be used to get or set the current length (in characters) of the string buffer. When setting the length, the new length is checked whether it lies within the maximum capacity (`MaxCapacity` (1969)) for the string buffer: if not, an `ERangeError` (1820) exception is raised. The capacity of the buffer will be adjusted so the buffer has enough capacity to accomodate the new length.

See also: `MaxCapacity` (1969), `Capacity` (1969), `ERangeError` (1820)

76.115.19 TUNICODESTRINGBUILDER.Capacity

Synopsis: Current capacity of the string buffer.

Declaration: `Property Capacity : Integer`

Visibility: public

Access: Read,Write

Description: `Capacity` can be used to get or set the capacity of the string buffer (in characters). When setting the capacity, the new length is checked whether it lies within the maximum capacity (`MaxCapacity` (1969)) for the string buffer: if not, an `ERangeError` (1820) exception is raised.

See also: `MaxCapacity` (1969), `Length` (1969), `ERangeError` (1820)

76.115.20 TUNICODESTRINGBUILDER.MaxCapacity

Synopsis: Maximum capacity of the string buffer.

Declaration: `Property MaxCapacity : Integer`

Visibility: public

Access: Read

Description: `MaxCapacity` returns the maximum capacity of the string buffer (in characters). It is set when the stringbuilder is created, in the constructor of the builder. The `Capacity` (1969) (and consequently, the `Length` (1969)) will never be allowed to exceed `MaxCapacity`.

See also: `Capacity` (1969), `Length` (1969), `ERangeError` (1820)

76.116 TUnicodeStringHelper

76.116.1 Method overview

Page	Method	Description
1972	Compare	
1972	CompareOrdinal	
1972	CompareText	
1975	CompareTo	
1975	Contains	
1972	Copy	
1976	CopyTo	
1976	CountChar	
1973	Create	
1976	DeQuotedString	
1973	EndsText	
1976	EndsWith	
1973	Equals	
1973	Format	
1976	GetHashCode	
1976	IndexOf	
1977	IndexOfAny	
1977	IndexOfAnyUnquoted	
1977	IndexOfUnquoted	
1977	Insert	
1978	IsDelimiter	
1978	IsEmpty	
1973	IsNullOrEmpty	
1973	IsNullOrWhiteSpace	
1974	Join	
1978	LastDelimiter	
1978	LastIndexOf	
1978	LastIndexOfAny	
1974	LowerCase	
1978	PadLeft	
1979	PadRight	
1974	Parse	
1979	QuotedString	
1979	Remove	
1979	Replace	
1979	Split	
1980	StartsWith	
1981	Substring	
1974	ToBoolean	
1981	ToCharArray	
1974	ToDouble	
1975	ToExtended	
1975	ToInt64	
1975	ToInteger	
1981	ToLower	
1981	ToLowerInvariant	
1975	ToSingle	
1981	ToUpper	
1981	ToUpperInvariant	
1981	Trim	
1982	TrimEnd	
1981	TrimLeft	1971
1982	TrimRight	
1982	TrimStart	
1975	UpperCase	

76.116.2 Property overview

Page	Properties	Access	Description
1982	Chars	r	
1982	Length	r	

76.116.3 TUnicodeStringHelper.Compare

Declaration: class function Compare(const A: UnicodeString; const B: UnicodeString)
 : Integer; Overload; Static
 class function Compare(const A: UnicodeString; const B: UnicodeString;
 IgnoreCase: Boolean) : Integer; Overload; Static
 class function Compare(const A: UnicodeString; const B: UnicodeString;
 Options: TCompareOptions) : Integer; Overload
 ; Static
 class function Compare(const A: UnicodeString; IndexA: SizeInt;
 const B: UnicodeString; IndexB: SizeInt;
 ALen: SizeInt) : Integer; Overload; Static
 class function Compare(const A: UnicodeString; IndexA: SizeInt;
 const B: UnicodeString; IndexB: SizeInt;
 ALen: SizeInt; IgnoreCase: Boolean) : Integer
 ; Overload; Static
 class function Compare(const A: UnicodeString; IndexA: SizeInt;
 const B: UnicodeString; IndexB: SizeInt;
 ALen: SizeInt; Options: TCompareOptions) : Integer
 ; Overload; Static

Visibility: public

76.116.4 TUnicodeStringHelper.CompareOrdinal

Declaration: class function CompareOrdinal(const A: UnicodeString;
 const B: UnicodeString) : Integer
 ; Overload; Static
 class function CompareOrdinal(const A: UnicodeString; IndexA: SizeInt;
 const B: UnicodeString; IndexB: SizeInt;
 ALen: SizeInt) : Integer; Overload
 ; Static

Visibility: public

76.116.5 TUnicodeStringHelper.CompareText

Declaration: class function CompareText(const A: UnicodeString;
 const B: UnicodeString) : Integer; Static

Visibility: public

76.116.6 TUnicodeStringHelper.Copy

Declaration: class function Copy(const Str: UnicodeString) : UnicodeString; Static

Visibility: public

76.116.13 TUnicodeStringHelper.Join

Declaration: class function Join(const Separator: UnicodeString;
 const Values: Array of const) : UnicodeString
 ; Overload; Static
 class function Join(const Separator: UnicodeString;
 const Values: Array of UnicodeString) : UnicodeString
 ; Overload; Static
 class function Join(const Separator: UnicodeString;
 const Values: Array of UnicodeString;
 StartIndex: SizeInt; ACount: SizeInt) : UnicodeString
 ; Overload; Static

Visibility: public

76.116.14 TUnicodeStringHelper.LowerCase

Declaration: class function LowerCase(const S: UnicodeString) : UnicodeString
 ; Overload; Static

Visibility: public

76.116.15 TUnicodeStringHelper.Parse

Declaration: class function Parse(const AValue: Boolean) : UnicodeString; Overload
 ; Static
 class function Parse(const AValue: Extended) : UnicodeString; Overload
 ; Static
 class function Parse(const AValue: Int64) : UnicodeString; Overload
 ; Static
 class function Parse(const AValue: Integer) : UnicodeString; Overload
 ; Static

Visibility: public

76.116.16 TUnicodeStringHelper.ToBoolean

Declaration: class function ToBoolean(const S: UnicodeString) : Boolean; Overload
 ; Static
 function ToBoolean : Boolean; Overload

Visibility: public

76.116.17 TUnicodeStringHelper.ToDouble

Declaration: class function ToDouble(const S: UnicodeString) : Double; Overload
 ; Static
 function ToDouble : Double; Overload

Visibility: public

76.116.25 TUnicodeStringHelper.CopyTo

[illegible]

Visibility: public

76.116.26 TUnicodeStringHelper.CountChar

Declaration: `function CountChar(const C: UnicodeChar) : SizeInt`

Visibility: public

76.116.27 TUnicodeStringHelper.DeQuotedString

```
Declaration: function DeQuotedString : UnicodeString; Overload
            function DeQuotedString(const AQuoteChar: UnicodeChar) : UnicodeString
                                ; Overload
```

Visibility: public

76.116.28 TUnicodeStringHelper.EndsWith

```
Declaration: function EndsWith(const AValue: UnicodeString) : Boolean; Overload
            function EndsWith(const AValue: UnicodeString; IgnoreCase: Boolean)
                        : Boolean; Overload
```

Visibility: public

76.116.29 TUnicodeStringHelper.GetHashCode

```
Declaration: function GetHashCode : Integer
```

Visibility: public

76.116.30 TUnicodeStringHelper.IndexOf

```
Declaration: function IndexOf(AValue: UnicodeChar) : SizeInt; Overload
            function IndexOf(const AValue: UnicodeString) : SizeInt; Overload
            function IndexOf(AValue: UnicodeChar; StartIndex: SizeInt) : SizeInt
                    ; Overload
            function IndexOf(const AValue: UnicodeString; StartIndex: SizeInt)
                    : SizeInt; Overload
            function IndexOf(AValue: UnicodeChar; StartIndex: SizeInt;
                    ACount: SizeInt) : SizeInt; Overload
            function IndexOf(const AValue: UnicodeString; StartIndex: SizeInt;
                    ACount: SizeInt) : SizeInt; Overload
```

Visibility: public

76.116.31 TUnicodeStringHelper.IndexOfUnQuoted

Declaration: function IndexOfUnQuoted(const AValue: UnicodeString;
 StartQuote: UnicodeChar; EndQuote: UnicodeChar;
 StartIndex: SizeInt) : SizeInt; Overload

Visibility: public

76.116.32 TUnicodeStringHelper.IndexOfAny

Declaration: function IndexOfAny(const AnyOf: Array of UnicodeChar) : SizeInt
 ; Overload
 function IndexOfAny(const AnyOf: Array of UnicodeChar;
 StartIndex: SizeInt) : SizeInt; Overload
 function IndexOfAny(const AnyOf: Array of UnicodeChar;
 StartIndex: SizeInt; ACount: SizeInt) : SizeInt
 ; Overload
 function IndexOfAny(const AnyOf: Array of UnicodeString) : SizeInt
 ; Overload
 function IndexOfAny(const AnyOf: Array of UnicodeString;
 StartIndex: SizeInt) : SizeInt; Overload
 function IndexOfAny(const AnyOf: Array of UnicodeString;
 StartIndex: SizeInt; ACount: SizeInt) : SizeInt
 ; Overload
 function IndexOfAny(const AnyOf: Array of UnicodeString;
 StartIndex: SizeInt; ACount: SizeInt;
 out AMatch: SizeInt) : SizeInt; Overload

Visibility: public

76.116.33 TUnicodeStringHelper.IndexOfAnyUnquoted

Declaration: function IndexOfAnyUnquoted(const AnyOf: Array of UnicodeChar;
 StartQuote: UnicodeChar;
 EndQuote: UnicodeChar) : SizeInt; Overload
 function IndexOfAnyUnquoted(const AnyOf: Array of UnicodeChar;
 StartQuote: UnicodeChar;
 EndQuote: UnicodeChar; StartIndex: SizeInt)
 : SizeInt; Overload
 function IndexOfAnyUnquoted(const AnyOf: Array of UnicodeChar;
 StartQuote: UnicodeChar;
 EndQuote: UnicodeChar; StartIndex: SizeInt;
 ACount: SizeInt) : SizeInt; Overload
 function IndexOfAnyUnquoted(const AnyOf: Array of UnicodeString;
 StartQuote: UnicodeChar;
 EndQuote: UnicodeChar; StartIndex: SizeInt;
 out Matched: SizeInt) : SizeInt; Overload

Visibility: public

76.116.34 TUnicodeStringHelper.Insert

Declaration: function Insert(StartIndex: SizeInt; const AValue: UnicodeString)
 : UnicodeString

Visibility: public

76.116.35 TUnicodeStringHelper.IsDelimiter

Declaration: function IsDelimiter(const Delimiters: UnicodeString; Index: SizeInt)
: Boolean

Visibility: public

76.116.36 TUnicodeStringHelper.IsEmpty

Declaration: function IsEmpty : Boolean

Visibility: public

76.116.37 TUnicodeStringHelper.LastDelimiter

Declaration: function LastDelimiter(const Delims: UnicodeString) : SizeInt

Visibility: public

76.116.38 TUnicodeStringHelper.LastIndexOf

Declaration: function LastIndexOf(AValue: UnicodeChar) : SizeInt; Overload
function LastIndexOf(const AValue: UnicodeString) : SizeInt; Overload
function LastIndexOf(AValue: UnicodeChar; AStartIndex: SizeInt)
: SizeInt; Overload
function LastIndexOf(const AValue: UnicodeString; AStartIndex: SizeInt)
: SizeInt; Overload
function LastIndexOf(AValue: UnicodeChar; AStartIndex: SizeInt;
ACount: SizeInt) : SizeInt; Overload
function LastIndexOf(const AValue: UnicodeString; AStartIndex: SizeInt;
ACount: SizeInt) : SizeInt; Overload

Visibility: public

76.116.39 TUnicodeStringHelper.LastIndexOfAny

Declaration: function LastIndexOfAny(const AnyOf: Array of UnicodeChar) : SizeInt
; Overload
function LastIndexOfAny(const AnyOf: Array of UnicodeChar;
AStartIndex: SizeInt) : SizeInt; Overload
function LastIndexOfAny(const AnyOf: Array of UnicodeChar;
AStartIndex: SizeInt; ACount: SizeInt) : SizeInt
; Overload

Visibility: public

76.116.40 TUnicodeStringHelper.PadLeft

Declaration: function PadLeft(ATotalWidth: SizeInt) : UnicodeString; Overload
function PadLeft(ATotalWidth: SizeInt; PaddingChar: UnicodeChar)
: UnicodeString; Overload

Visibility: public

76.116.41 TUnicodeStringHelper.PadRight

```
Declaration: function PadRight(ATotalWidth: SizeInt) : UnicodeString; Overload
function PadRight(ATotalWidth: SizeInt; PaddingChar: UnicodeChar)
: UnicodeString; Overload
```

Visibility: public

76.116.42 TUnicodeStringHelper.QuotedString

```
Declaration: function QuotedString : UnicodeString; Overload
            function QuotedString(const AQuoteChar: UnicodeChar) : UnicodeString
                ; Overload
```

Visibility: public

76.116.43 TUnicodeStringHelper.Remove

```
Declaration: function Remove(StartIndex: SizeInt) : UnicodeString; Overload
            function Remove(StartIndex: SizeInt; ACount: SizeInt) : UnicodeString
                    ; Overload
```

Visibility: public

76.116.44 TUnicodeStringHelper.Replace

```

Declaration: function Replace(OldChar: UnicodeChar; NewChar: UnicodeChar)
                        : UnicodeString; Overload
function Replace(OldChar: UnicodeChar; NewChar: UnicodeChar;
                ReplaceFlags: TReplaceFlags) : UnicodeString; Overload
function Replace(const OldValue: UnicodeString;
                const NewValue: UnicodeString) : UnicodeString
                ; Overload
function Replace(const OldValue: UnicodeString;
                const NewValue: UnicodeString;
                ReplaceFlags: TReplaceFlags) : UnicodeString; Overload

```

Visibility: public

76.116.45 TUnicodeStringHelper.Split

```

Declaration: function Split(const Separators: Array of UnicodeChar)
                        : TUnicodeStringArray;  Overload
function Split(const Separators: Array of UnicodeChar; ACount: SizeInt)
                        : TUnicodeStringArray;  Overload
function Split(const Separators: Array of UnicodeChar;
                Options: TStringSplitOptions) : TUnicodeStringArray
                        ;  Overload
function Split(const Separators: Array of UnicodeChar; ACount: SizeInt;
                Options: TStringSplitOptions) : TUnicodeStringArray

```

```

        ; Overload
function Split(const Separators: Array of UnicodeString)
        : TUnicodeStringArray; Overload
function Split(const Separators: Array of UnicodeString;
        ACount: SizeInt) : TUnicodeStringArray; Overload
function Split(const Separators: Array of UnicodeString;
        Options: TStringSplitOptions) : TUnicodeStringArray
        ; Overload
function Split(const Separators: Array of UnicodeString;
        ACount: SizeInt; Options: TStringSplitOptions)
        : TUnicodeStringArray; Overload
function Split(const Separators: Array of UnicodeChar;
        AQuote: UnicodeChar) : TUnicodeStringArray; Overload
function Split(const Separators: Array of UnicodeChar;
        AQuoteStart: UnicodeChar; AQuoteEnd: UnicodeChar)
        : TUnicodeStringArray; Overload
function Split(const Separators: Array of UnicodeChar;
        AQuoteStart: UnicodeChar; AQuoteEnd: UnicodeChar;
        Options: TStringSplitOptions) : TUnicodeStringArray
        ; Overload
function Split(const Separators: Array of UnicodeChar;
        AQuoteStart: UnicodeChar; AQuoteEnd: UnicodeChar;
        ACount: SizeInt) : TUnicodeStringArray; Overload
function Split(const Separators: Array of UnicodeChar;
        AQuoteStart: UnicodeChar; AQuoteEnd: UnicodeChar;
        ACount: SizeInt; Options: TStringSplitOptions)
        : TUnicodeStringArray; Overload
function Split(const Separators: Array of UnicodeString;
        AQuote: UnicodeChar) : TUnicodeStringArray; Overload
function Split(const Separators: Array of UnicodeString;
        AQuoteStart: UnicodeChar; AQuoteEnd: UnicodeChar)
        : TUnicodeStringArray; Overload
function Split(const Separators: Array of UnicodeString;
        AQuoteStart: UnicodeChar; AQuoteEnd: UnicodeChar;
        Options: TStringSplitOptions) : TUnicodeStringArray
        ; Overload
function Split(const Separators: Array of UnicodeString;
        AQuoteStart: UnicodeChar; AQuoteEnd: UnicodeChar;
        ACount: SizeInt) : TUnicodeStringArray; Overload
function Split(const Separators: Array of UnicodeString;
        AQuoteStart: UnicodeChar; AQuoteEnd: UnicodeChar;
        ACount: SizeInt; Options: TStringSplitOptions)
        : TUnicodeStringArray; Overload

```

Visibility: public

76.116.46 TUnicodeStringHelper.StartsWith

Declaration: function StartsWith(const AValue: UnicodeString) : Boolean; Overload
function StartsWith(const AValue: UnicodeString; IgnoreCase: Boolean)
: Boolean; Overload

Visibility: public

76.116.47 TUnicodeStringHelper.Substring

Declaration: function Substring(AStartIndex: SizeInt) : UnicodeString; Overload
function Substring(AStartIndex: SizeInt; ALen: SizeInt) : UnicodeString
; Overload

Visibility: public

76.116.48 TUnicodeStringHelper.ToCharArray

Declaration: function ToCharArray : TCharArray; Overload
function ToCharArray(AStartIndex: SizeInt; ALen: SizeInt) : TCharArray
; Overload

Visibility: public

76.116.49 TUnicodeStringHelper.ToLower

Declaration: function ToLower : UnicodeString; Overload

Visibility: public

76.116.50 TUnicodeStringHelper.ToLowerInvariant

Declaration: function ToLowerInvariant : UnicodeString

Visibility: public

76.116.51 TUnicodeStringHelper.ToUpper

Declaration: function ToUpper : UnicodeString; Overload

Visibility: public

76.116.52 TUnicodeStringHelper.ToUpperInvariant

Declaration: function ToUpperInvariant : UnicodeString

Visibility: public

76.116.53 TUnicodeStringHelper.Trim

Declaration: function Trim : UnicodeString; Overload
function Trim(const ATrimChars: Array of UnicodeChar) : UnicodeString
; Overload

Visibility: public

76.116.54 TUnicodeStringHelper.TrimLeft

Declaration: function TrimLeft : UnicodeString; Overload
function TrimLeft(const ATrimChars: Array of UnicodeChar)
: UnicodeString; Overload

Visibility: public

76.116.55 TUnicodeStringHelper.TrimRight

Declaration: `function TrimRight : UnicodeString; Overload`
`function TrimRight(const ATrimChars: Array of UnicodeChar)`
`: UnicodeString; Overload`

Visibility: public

76.116.56 TUnicodeStringHelper.TrimEnd

Declaration: `function TrimEnd(const ATrimChars: Array of UnicodeChar) : UnicodeString`

Visibility: public

76.116.57 TUnicodeStringHelper.TrimStart

Declaration: `function TrimStart(const ATrimChars: Array of UnicodeChar)`
`: UnicodeString`

Visibility: public

76.116.58 TUnicodeStringHelper.Chars

Declaration: `Property Chars[AIndex: SizeInt]: UnicodeChar`

Visibility: public

Access: Read

76.116.59 TUnicodeStringHelper.Length

Declaration: `Property &Length : SizeInt`

Visibility: public

Access: Read

76.117 TUTF7Encoding**76.117.1 Description**

TUTF7Encoding is the encoding class used to represent the UTF7 encoding. This encoding is not so often used, and the class is provided mostly for completeness.

See also: TUnicodeEncoding ([1961](#)), TUTF8Encoding ([1983](#)), TMBCSEncoding ([1904](#)), TBigendianUnicodeEncoding ([1846](#))

76.117.2 Method overview

Page	Method	Description
1983	Clone	Clone a TUTF7Encoding instance.
1983	Create	Create a new instance of the TUTF7Encoding class.
1983	GetMaxByteCount	Return max number of bytes needed to represent a string.
1983	GetMaxCharCount	Return max number of characters that can be represented by an array of bytes.

76.117.3 TUTF7Encoding.Create

Synopsis: Create a new instance of the TUTF7Encoding class.

Declaration: constructor Create; Override

Visibility: public

Description: Create creates a new instance of the TUTF7Encoding class and sets the codepage to CP_UTF7.

See also: TEncoding.CodePage ([1875](#)), TEncoding ([1871](#))

76.117.4 TUTF7Encoding.Clone

Synopsis: Clone a TUTF7Encoding instance.

Declaration: function Clone : TEncoding; Override

Visibility: public

Description: Clone overrides TEncoding.Clone ([1872](#)) to provide a clone of the TUTF7Encoding instance.

See also: TEncoding.Clone ([1872](#))

76.117.5 TUTF7Encoding.GetMaxByteCount

Synopsis: Return max number of bytes needed to represent a string.

Declaration: function GetMaxByteCount(CharCount: Integer) : Integer; Override

Visibility: public

Description: GetMaxByteCount overrides TEncoding.GetMaxByteCount ([1874](#)) to return the maximum number of bytes needed to represent a string.

See also: TEncoding.GetMaxByteCount ([1874](#))

76.117.6 TUTF7Encoding.GetMaxCharCount

Synopsis: Return max number of characters that can be represented by an array of bytes.

Declaration: function GetMaxCharCount(ByteCount: Integer) : Integer; Override

Visibility: public

Description: GetMaxCharCount overrides TEncoding.GetMaxCharCount ([1874](#)) to return the maximum number of bytes needed to represent a string.

See also: TEncoding.GetMaxCharCount ([1874](#))

76.118 TUTF8Encoding**76.118.1 Description**

TUTF8Encoding is the encoding class used to represent the UTF-8 encoding.

See also: TUnicodeEncoding ([1961](#)), TUTF7Encoding ([1982](#)), TMBCSEncoding ([1904](#)), TBigendianUnicodeEncoding ([1846](#))

76.118.2 Method overview

Page	Method	Description
1984	Clone	Clone a <code>TUTF8Encoding</code> instance.
1984	Create	Create a new instance of the <code>TUTF8Encoding</code> class.
1984	GetMaxByteCount	Return max number of bytes needed to represent a string.
1984	GetMaxCharCount	Return max number of characters that can be represented by an array of bytes.
1985	GetPreamble	Return BOM marker bytes.

76.118.3 TUTF8Encoding.Create

Synopsis: Create a new instance of the `TUTF8Encoding` class.

Declaration: `constructor Create; Override`

Visibility: `public`

Description: `Create` creates a new instance of the `TUTF8Encoding` class and sets the codepage to `CP_UTF8`.

See also: `TEncoding.CodePage` ([1875](#)), `TEncoding` ([1871](#))

76.118.4 TUTF8Encoding.Clone

Synopsis: Clone a `TUTF8Encoding` instance.

Declaration: `function Clone : TEncoding; Override`

Visibility: `public`

Description: `Clone` overrides `TEncoding.Clone` ([1872](#)) to provide a clone of the `TUTF8Encoding` instance.

See also: `TEncoding.Clone` ([1872](#))

76.118.5 TUTF8Encoding.GetMaxByteCount

Synopsis: Return max number of bytes needed to represent a string.

Declaration: `function GetMaxByteCount (CharCount: Integer) : Integer; Override`

Visibility: `public`

Description: `GetMaxByteCount` overrides `TEncoding.GetMaxByteCount` ([1874](#)) to return the maximum number of bytes needed to represent a string.

See also: `TEncoding.GetMaxByteCount` ([1874](#))

76.118.6 TUTF8Encoding.GetMaxCharCount

Synopsis: Return max number of characters that can be represented by an array of bytes.

Declaration: `function GetMaxCharCount (ByteCount: Integer) : Integer; Override`

Visibility: `public`

Description: `GetMaxCharCount` overrides `TEncoding.GetMaxCharCount` ([1874](#)) to return the maximum number of bytes needed to represent a string.

See also: `TEncoding.GetMaxCharCount` ([1874](#))

76.118.7 TUTF8Encoding.GetPreamble

Synopsis: Return BOM marker bytes.

Declaration: `function GetPreamble : TBytes; Override`

Visibility: `public`

Description: `GetPreamble` overrides `TEncoding.GetPreamble` ([1874](#)) to return the 3 UTF8 BOM Marker bytes (\$EF,\$BB,\$BF).

See also: `TEncoding.GetPreamble` ([1874](#))

76.119 TWideStringHelper

76.119.1 Method overview

Page	Method	Description
1988	Compare	
1988	CompareOrdinal	
1988	CompareText	
1991	CompareTo	
1991	Contains	
1988	Copy	
1992	CopyTo	
1992	CountChar	
1989	Create	
1992	DeQuotedString	
1989	EndsText	
1992	EndsWith	
1989	Equals	
1989	Format	
1992	GetHashCode	
1992	IndexOf	
1993	IndexOfAny	
1993	IndexOfAnyUnquoted	
1993	IndexOfUnquoted	
1993	Insert	
1994	IsDelimiter	
1994	IsEmpty	
1989	IsNullOrEmpty	
1989	IsNullOrWhiteSpace	
1990	Join	
1994	LastDelimiter	
1994	LastIndexOf	
1994	LastIndexOfAny	
1990	LowerCase	
1994	PadLeft	
1995	PadRight	
1990	Parse	
1995	QuotedString	
1995	Remove	
1995	Replace	
1995	Split	
1996	StartsWith	
1996	Substring	
1990	ToBoolean	
1997	ToCharArray	
1990	ToDouble	
1991	ToExtended	
1991	ToInt64	
1991	ToInteger	
1997	ToLower	
1997	ToLowerInvariant	
1991	ToSingle	
1997	ToUpper	
1997	ToUpperInvariant	
1997	Trim	
1998	TrimEnd	
1997	TrimLeft	1987
1998	TrimRight	
1998	TrimStart	
1991	UpperCase	

76.119.2 Property overview

Page	Properties	Access	Description
1998	Chars	r	
1998	Length	r	

76.119.3 TWideStringHelper.Compare

```

Declaration: class function Compare(const A: WideString; const B: WideString)
              : Integer; Overload; Static
class function Compare(const A: WideString; const B: WideString;
              IgnoreCase: Boolean) : Integer; Overload; Static
class function Compare(const A: WideString; const B: WideString;
              Options: TCompareOptions) : Integer; Overload
              ; Static
class function Compare(const A: WideString; IndexA: SizeInt;
              const B: WideString; IndexB: SizeInt;
              ALen: SizeInt) : Integer; Overload; Static
class function Compare(const A: WideString; IndexA: SizeInt;
              const B: WideString; IndexB: SizeInt;
              ALen: SizeInt; IgnoreCase: Boolean) : Integer
              ; Overload; Static
class function Compare(const A: WideString; IndexA: SizeInt;
              const B: WideString; IndexB: SizeInt;
              ALen: SizeInt; Options: TCompareOptions) : Integer
              ; Overload; Static

```

Visibility: public

76.119.4 TWideStringHelper.CompareOrdinal

```
Declaration: class function CompareOrdinal(const A: WideString; const B: WideString)
            : Integer; Overload; Static
            class function CompareOrdinal(const A: WideString; IndexA: SizeInt;
            const B: WideString; IndexB: SizeInt;
            ALen: SizeInt) : Integer; Overload
            ; Static
```

Visibility: public

76.119.5 TWideStringHelper.CompareText

```
Declaration: class function CompareText(const A: WideString; const B: WideString)
              : Integer; Static
```

Visibility: public

76.119.6 TWideStringHelper.Copy

```
Declaration: class function Copy(const Str: WideString) : WideString; Static
```

Visibility: public

76.119.7 TWideStringHelper.Create

```
Declaration: class function Create(AChar: WideChar; ACount: SizeInt) : WideString
                ; Overload; Static
class function Create(const AValue: Array of WideChar) : WideString
                ; Overload; Static
class function Create(const AValue: Array of WideChar;
                StartIndex: SizeInt; ALen: SizeInt) : WideString
                ; Overload; Static
```

Visibility: public

76.119.8 TWideStringHelper.EndsText

[illegible]

Visibility: public

76.119.9 TWideStringHelper.Equals

```
Declaration: class function Equals(const a: WideString; const b: WideString)
                : Boolean; Overload; Static
function Equals(const AValue: WideString; IgnoreCase: Boolean) : Boolean
                : Overload
```

Visibility: public

76.119.10 TWideStringHelper.Format

```
Declaration: class function Format(const AFormat: WideString;
                                const args: Array of const) : WideString; Overload
                                ; Static
                                function Format(const args: Array of const) : WideString; Overload
```

Visibility: public

76.119.11 TWideStringHelper.IsNullOrEmpty

```
Declaration: class function IsNullOrEmpty(const AValue: WideString) : Boolean
; Static
```

Visibility: public

76.119.12 TWideStringHelper.IsNullOrWhiteSpace

```
Declaration: class function IsNullOrWhiteSpace(const AValue: WideString) : Boolean
; Static
```

Visibility: public

76.119.13 TWideStringHelper.Join

Declaration: class function Join(const Separator: WideString;
 const Values: Array of const) : WideString; Overload
 ; Static
 class function Join(const Separator: WideString;
 const Values: Array of WideString) : WideString
 ; Overload; Static
 class function Join(const Separator: WideString;
 const Values: Array of WideString;
 StartIndex: SizeInt; ACount: SizeInt) : WideString
 ; Overload; Static

Visibility: public

76.119.14 TWideStringHelper.LowerCase

Declaration: class function LowerCase(const S: WideString) : WideString; Overload
 ; Static

Visibility: public

76.119.15 TWideStringHelper.Parse

Declaration: class function Parse(const AValue: Boolean) : WideString; Overload
 ; Static
 class function Parse(const AValue: Extended) : WideString; Overload
 ; Static
 class function Parse(const AValue: Int64) : WideString; Overload
 ; Static
 class function Parse(const AValue: Integer) : WideString; Overload
 ; Static

Visibility: public

76.119.16 TWideStringHelper.ToBoolean

Declaration: class function ToBoolean(const S: WideString) : Boolean; Overload
 ; Static
 function ToBoolean : Boolean; Overload

Visibility: public

76.119.17 TWideStringHelper.ToDouble

Declaration: class function ToDouble(const S: WideString) : Double; Overload
 ; Static
 function ToDouble : Double; Overload

Visibility: public

76.119.25 TWideStringHelper.CopyTo

Declaration: procedure CopyTo(SourceIndex: SizeInt;
 var destination: Array of WideChar;
 DestinationIndex: SizeInt; ACount: SizeInt)

Visibility: public

76.119.26 TWideStringHelper.CountChar

Declaration: function CountChar(const C: WideChar) : SizeInt

Visibility: public

76.119.27 TWideStringHelper.DeQuotedString

Declaration: function DeQuotedString : WideString; Overload
 function DeQuotedString(const AQuoteChar: WideChar) : WideString
 ; Overload

Visibility: public

76.119.28 TWideStringHelper.EndsWith

Declaration: function EndsWith(const AValue: WideString) : Boolean; Overload
 function EndsWith(const AValue: WideString; IgnoreCase: Boolean)
 : Boolean; Overload

Visibility: public

76.119.29 TWideStringHelper.GetHashCode

Declaration: function GetHashCode : Integer

Visibility: public

76.119.30 TWideStringHelper.IndexOf

Declaration: function IndexOf(AValue: WideChar) : SizeInt; Overload
 function IndexOf(const AValue: WideString) : SizeInt; Overload
 function IndexOf(AValue: WideChar; StartIndex: SizeInt) : SizeInt
 ; Overload
 function IndexOf(const AValue: WideString; StartIndex: SizeInt)
 : SizeInt; Overload
 function IndexOf(AValue: WideChar; StartIndex: SizeInt; ACount: SizeInt)
 : SizeInt; Overload
 function IndexOf(const AValue: WideString; StartIndex: SizeInt;
 ACount: SizeInt) : SizeInt; Overload

Visibility: public

76.119.31 TWideStringHelper.IndexOfUnQuoted

Declaration: function IndexOfUnQuoted(const AValue: WideString;
 StartQuote: WideChar; EndQuote: WideChar;
 StartIndex: SizeInt) : SizeInt; Overload

Visibility: public

76.119.32 TWideStringHelper.IndexOfAny

Declaration: function IndexOfAny(const AnyOf: Array of WideChar) : SizeInt; Overload
 function IndexOfAny(const AnyOf: Array of WideChar; StartIndex: SizeInt)
 : SizeInt; Overload
 function IndexOfAny(const AnyOf: Array of WideChar;
 StartIndex: SizeInt; ACount: SizeInt) : SizeInt
 ; Overload
 function IndexOfAny(const AnyOf: Array of WideString) : SizeInt
 ; Overload
 function IndexOfAny(const AnyOf: Array of WideString;
 StartIndex: SizeInt) : SizeInt; Overload
 function IndexOfAny(const AnyOf: Array of WideString;
 StartIndex: SizeInt; ACount: SizeInt) : SizeInt
 ; Overload
 function IndexOfAny(const AnyOf: Array of WideString;
 StartIndex: SizeInt; ACount: SizeInt;
 out AMatch: SizeInt) : SizeInt; Overload

Visibility: public

76.119.33 TWideStringHelper.IndexOfAnyUnquoted

Declaration: function IndexOfAnyUnquoted(const AnyOf: Array of WideChar;
 StartQuote: WideChar; EndQuote: WideChar)
 : SizeInt; Overload
 function IndexOfAnyUnquoted(const AnyOf: Array of WideChar;
 StartQuote: WideChar; EndQuote: WideChar;
 StartIndex: SizeInt) : SizeInt; Overload
 function IndexOfAnyUnquoted(const AnyOf: Array of WideChar;
 StartQuote: WideChar; EndQuote: WideChar;
 StartIndex: SizeInt; ACount: SizeInt)
 : SizeInt; Overload
 function IndexOfAnyUnquoted(const AnyOf: Array of WideString;
 StartQuote: WideChar; EndQuote: WideChar;
 StartIndex: SizeInt; out Matched: SizeInt)
 : SizeInt; Overload

Visibility: public

76.119.34 TWideStringHelper.Insert

Declaration: function Insert(StartIndex: SizeInt; const AValue: WideString)
 : WideString

Visibility: public

76.119.35 TWideStringHelper.IsDelimiter

Declaration: function IsDelimiter(const Delimiters: WideString; Index: SizeInt)
: Boolean

Visibility: public

76.119.36 TWideStringHelper.IsEmpty

Declaration: function IsEmpty : Boolean

Visibility: public

76.119.37 TWideStringHelper.LastDelimiter

Declaration: function LastDelimiter(const Delims: WideString) : SizeInt

Visibility: public

76.119.38 TWideStringHelper.LastIndexOf

Declaration: function LastIndexOf(AValue: WideChar) : SizeInt; Overload
function LastIndexOf(const AValue: WideString) : SizeInt; Overload
function LastIndexOf(AValue: WideChar; AStartIndex: SizeInt) : SizeInt
; Overload
function LastIndexOf(const AValue: WideString; AStartIndex: SizeInt)
: SizeInt; Overload
function LastIndexOf(AValue: WideChar; AStartIndex: SizeInt;
ACount: SizeInt) : SizeInt; Overload
function LastIndexOf(const AValue: WideString; AStartIndex: SizeInt;
ACount: SizeInt) : SizeInt; Overload

Visibility: public

76.119.39 TWideStringHelper.LastIndexOfAny

Declaration: function LastIndexOfAny(const AnyOf: Array of WideChar) : SizeInt
; Overload
function LastIndexOfAny(const AnyOf: Array of WideChar;
AStartIndex: SizeInt) : SizeInt; Overload
function LastIndexOfAny(const AnyOf: Array of WideChar;
AStartIndex: SizeInt; ACount: SizeInt) : SizeInt
; Overload

Visibility: public

76.119.40 TWideStringHelper.PadLeft

Declaration: function PadLeft(ATotalWidth: SizeInt) : WideString; Overload
function PadLeft(ATotalWidth: SizeInt; PaddingChar: WideChar)
: WideString; Overload

Visibility: public

76.119.41 TWideStringHelper.PadRight

Declaration: function PadRight(ATotalWidth: SizeInt) : WideString; Overload
function PadRight(ATotalWidth: SizeInt; PaddingChar: WideChar)
: WideString; Overload

Visibility: public

76.119.42 TWideStringHelper.QuotedString

Declaration: function QuotedString : WideString; Overload
function QuotedString(const AQuoteChar: WideChar) : WideString
; Overload

Visibility: public

76.119.43 TWideStringHelper.Remove

Declaration: function Remove(StartIndex: SizeInt) : WideString; Overload
function Remove(StartIndex: SizeInt; ACount: SizeInt) : WideString
; Overload

Visibility: public

76.119.44 TWideStringHelper.Replace

Declaration: function Replace(OldChar: WideChar; NewChar: WideChar) : WideString
; Overload
function Replace(OldChar: WideChar; NewChar: WideChar;
ReplaceFlags: TReplaceFlags) : WideString; Overload
function Replace(const OldValue: WideString; const NewValue: WideString)
: WideString; Overload
function Replace(const OldValue: WideString;
const NewValue: WideString; ReplaceFlags: TReplaceFlags)
: WideString; Overload

Visibility: public

76.119.45 TWideStringHelper.Split

Declaration: function Split(const Separators: Array of WideChar) : TWideStringArray
; Overload
function Split(const Separators: Array of WideChar; ACount: SizeInt)
: TWideStringArray; Overload
function Split(const Separators: Array of WideChar;
Options: TStringSplitOptions) : TWideStringArray
; Overload
function Split(const Separators: Array of WideChar; ACount: SizeInt;
Options: TStringSplitOptions) : TWideStringArray
; Overload
function Split(const Separators: Array of WideString) : TWideStringArray
; Overload
function Split(const Separators: Array of WideString; ACount: SizeInt)
: TWideStringArray; Overload

```

function Split(const Separators: Array of WideString;
               Options: TStringSplitOptions) : TWideStringArray
    ; Overload
function Split(const Separators: Array of WideString; ACount: SizeInt;
               Options: TStringSplitOptions) : TWideStringArray
    ; Overload
function Split(const Separators: Array of WideChar; AQuote: WideChar)
    : TWideStringArray; Overload
function Split(const Separators: Array of WideChar;
               AQuoteStart: WideChar; AQuoteEnd: WideChar)
    : TWideStringArray; Overload
function Split(const Separators: Array of WideChar;
               AQuoteStart: WideChar; AQuoteEnd: WideChar;
               Options: TStringSplitOptions) : TWideStringArray
    ; Overload
function Split(const Separators: Array of WideChar;
               AQuoteStart: WideChar; AQuoteEnd: WideChar;
               ACount: SizeInt) : TWideStringArray; Overload
function Split(const Separators: Array of WideChar;
               AQuoteStart: WideChar; AQuoteEnd: WideChar;
               ACount: SizeInt; Options: TStringSplitOptions)
    : TWideStringArray; Overload
function Split(const Separators: Array of WideString; AQuote: WideChar)
    : TWideStringArray; Overload
function Split(const Separators: Array of WideString;
               AQuoteStart: WideChar; AQuoteEnd: WideChar)
    : TWideStringArray; Overload
function Split(const Separators: Array of WideString;
               AQuoteStart: WideChar; AQuoteEnd: WideChar;
               Options: TStringSplitOptions) : TWideStringArray
    ; Overload
function Split(const Separators: Array of WideString;
               AQuoteStart: WideChar; AQuoteEnd: WideChar;
               ACount: SizeInt) : TWideStringArray; Overload
function Split(const Separators: Array of WideString;
               AQuoteStart: WideChar; AQuoteEnd: WideChar;
               ACount: SizeInt; Options: TStringSplitOptions)
    : TWideStringArray; Overload

```

Visibility: public

76.119.46 TWideStringHelper.StartsWith

Declaration: function StartsWith(const AValue: WideString) : Boolean; Overload
 function StartsWith(const AValue: WideString; IgnoreCase: Boolean)
 : Boolean; Overload

Visibility: public

76.119.47 TWideStringHelper.Substring

Declaration: function Substring(AStartIndex: SizeInt) : WideString; Overload
 function Substring(AStartIndex: SizeInt; ALen: SizeInt) : WideString

; Overload

Visibility: public

76.119.48 TWideStringHelper.ToCharArray

Declaration: function ToCharArray : TCharArray; Overload
function ToCharArray(AStartIndex: SizeInt; ALen: SizeInt) : TCharArray
; Overload

Visibility: public

76.119.49 TWideStringHelper.ToLower

Declaration: function ToLower : WideString; Overload

Visibility: public

76.119.50 TWideStringHelper.ToLowerInvariant

Declaration: function ToLowerInvariant : WideString

Visibility: public

76.119.51 TWideStringHelper.ToUpper

Declaration: function ToUpper : WideString; Overload

Visibility: public

76.119.52 TWideStringHelper.ToUpperInvariant

Declaration: function ToUpperInvariant : WideString

Visibility: public

76.119.53 TWideStringHelper.Trim

Declaration: function Trim : WideString; Overload
function Trim(const ATrimChars: Array of WideChar) : WideString
; Overload

Visibility: public

76.119.54 TWideStringHelper.TrimLeft

Declaration: function TrimLeft : WideString; Overload
function TrimLeft(const ATrimChars: Array of WideChar) : WideString
; Overload

Visibility: public

76.119.55 TWideStringHelper.TrimRight

Declaration: `function TrimRight : WideString; Overload`
`function TrimRight(const ATrimChars: Array of WideChar) : WideString`
`; Overload`

Visibility: public

76.119.56 TWideStringHelper.TrimEnd

Declaration: `function TrimEnd(const ATrimChars: Array of WideChar) : WideString`

Visibility: public

76.119.57 TWideStringHelper.TrimStart

Declaration: `function TrimStart(const ATrimChars: Array of WideChar) : WideString`

Visibility: public

76.119.58 TWideStringHelper.Chars

Declaration: `Property Chars[AIndex: SizeInt]: WideChar`

Visibility: public

Access: Read

76.119.59 TWideStringHelper.Length

Declaration: `Property &Length : SizeInt`

Visibility: public

Access: Read

76.120 TWordBoolHelper

76.120.1 Description

TWordBoolHelper is a helper type for the WordBool type. It contains mostly conversion routines to and from other types.

See also: TStringHelper ([1609](#)), TShortIntHelper ([1928](#)), TSmallIntHelper ([1955](#)), TWordHelper ([2000](#)), TCardinalHelper ([1857](#)), TIntegerHelper ([1893](#)), TInt64Helper ([1886](#)), TQWordHelper ([1921](#)), TNativeIntHelper ([1908](#)), TByteHelper ([1851](#)), TByteBoolHelper ([1849](#)), TWordBoolHelper ([1998](#)), TLongBoolHelper ([1899](#))

76.120.2 Method overview

Page	Method	Description
1999	<code>Parse</code>	Convert string value to WordBool value.
1999	<code>Size</code>	Return the size (in bytes) of the.
2000	<code>ToInteger</code>	Convert to an integer value.
1999	<code>ToString</code>	Convert a WordBool value to string.
2000	<code>TryToParse</code>	Try to convert a string to a WordBool value.

76.120.3 TWordBoolHelper.Parse

Synopsis: Convert string value to WordBool value.

Declaration: `class function Parse(const S: string) : Boolean; Static`

Visibility: `public`

Description: `Parse` attempts to convert the string `S` to a WordBool value. It uses the `StrToBool` ([1768](#)) function to perform the conversion.

Errors: If `S` does not contain a valid string representation, then an `EConvertError` ([1812](#)) exception is raised.

See also: `TWordBoolHelper.TryToParse` ([2000](#)), `TWordBoolHelper.ToString` ([1999](#)), `TWordBoolHelper.ToInteger` ([2000](#))

76.120.4 TWordBoolHelper.Size

Synopsis: Return the size (in bytes) of the.

Declaration: `class function Size : Integer; Static`

Visibility: `public`

Description: `Size` returns the size (in bytes) of the WordBool value. This is equivalent to `SizeOf(WordBool)`.

See also: `SizeOf` ([1546](#))

76.120.5 TWordBoolHelper.ToString

Synopsis: Convert a WordBool value to string.

Declaration: `class function ToString(const AValue: Boolean;
 UseBoolStrs: TUseBoolStrs) : string; Overload
 ; Static
function ToString(UseBoolStrs: TUseBoolStrs) : string; Overload`

Visibility: `public`

Description: `ToString` will, in the class method version, convert the `AValue` WordBool to a string representation. In the function method version the WordBool value itself (`Self`) will be converted.

If the `UseBoolStrs` parameter equals `TUseBoolStrs.True`, then the string representation will use the WordBool strings `BoolStrs` ([1609](#)). The default value for `UseBoolStrs` is `TUseBoolStrs.False`.

The conversion is done using the `BoolToStr` ([1664](#)) function.

See also: `BoolStrs` ([1609](#)), `BoolToStr` ([1664](#))

76.120.6 TWordBoolHelper.TryToParse

Synopsis: Try to convert a string to a WordBool value.

Declaration: `class function TryToParse(const S: string; out AValue: Boolean)
: Boolean; Static`

Visibility: public

Description: `TryToParse` will attempt to convert the string `S` to a `WordBool` value. If the attempt is successful, `True` is returned, and the actual value is returned in `AValue`. If the attempt failed, `False` is returned.

See also: `TWordBoolHelper.Parse` ([1999](#)), `TWordBoolHelper.ToString` ([1999](#))

76.120.7 TWordBoolHelper.ToInteger

Synopsis: Convert to an integer value.

Declaration: `function ToInteger : Integer`

Visibility: public

Description: `ToInteger` will return the `WordBool` value, typecasted to `Integer`.

See also: `TWordBoolHelper.ToString` ([1999](#))

76.121 TWordHelper

76.121.1 Description

`TWordHelper` contains some auxiliary routines for a Word-typed ordinal value. It consists mainly of conversion routines to and from other types.

See also: `TStringHelper` ([1609](#)), `TShortIntHelper` ([1928](#)), `TSmallIntHelper` ([1955](#)), `TByteHelper` ([1851](#)), `TCardinalHelper` ([1857](#)), `TIntegerHelper` ([1893](#)), `TInt64Helper` ([1886](#)), `TQWordHelper` ([1921](#)), `TNativeIntHelper` ([1908](#)), `TNativeUIntHelper` ([1915](#))

76.121.2 Method overview

Page	Method	Description
2005	Clear	Set value to 0.
2004	ClearBit	Set bit to 0.
2005	HighestSetBitPos	Return the position of the leftmost bit set.
2005	LowestSetBitPos	Return the position of the rightmost bit set.
2001	Parse	Convert from a string.
2004	SetBit	Set bit to 1.
2005	SetBitsCount	Count number of bits set.
2001	Size	Size, in bytes, of the Word value.
2004	TestBit	Check bit.
2003	ToBinString	Convert to a binary string representation.
2002	ToBoolean	Convert to a boolean value.
2002	ToDouble	Convert to a double-sized floating point value.
2003	ToExtended	Convert to an extended-sized floating point value.
2004	ToggleBit	Invert bit.
2003	ToHexString	Convert to a hexadecimal string representation.
2003	ToSingle	Convert to a single-sized floating point value.
2002	ToString	Convert the value to string.
2002	TryParse	Try to convert a string to a Word, report success or failure.

76.121.3 Property overview

Page	Properties	Access	Description
2006	Bits	rw	Read or write a bit.
2006	Bytes	rw	Read or write a byte.
2006	Nibbles	rw	Read or write a nibble.

76.121.4 TWordHelper.Parse

Synopsis: Convert from a string.

Declaration: `class function Parse(const AString: string) : Word; Static`

Visibility: public

Description: `Parse` will attempt to convert the string `AString` to a `Word` value. It uses the `StrToInt` ([1773](#)) function to perform the conversion, so no localization is taken into account.

Errors: If the string does not contain a valid `Word` value, an `EConvertError` ([1812](#)) exception is raised.

See also: `TWordHelper.ToString` ([2002](#)), `TWordHelper.TryParse` ([2002](#)), `StrToInt` ([1773](#))

76.121.5 TWordHelper.Size

Synopsis: Size, in bytes, of the `Word` value.

Declaration: `class function Size : Integer; Static`

Visibility: public

Description: `Size` returns the size (in Words) of the `Word` value. This is equivalent to `SizeOf(Word)`.

Errors: None.

See also: `SizeOf` ([1546](#))

76.121.6 TWordHelper.ToString

Synopsis: Convert the value to string.

Declaration: `class function ToString(const AValue: Word) : string; Overload; Static
function ToString : string; Overload`

Visibility: public

Description: `ToString` will, in the class function variant of this method, convert `AValue` to a string representation. In the regular method overloaded version of `ToString`, the `Word` value itself is used. The `IntToStr` ([1738](#)) function is used to do the conversion.

See also: `TWordHelper.Parse` ([2001](#)), `IntToStr` ([1738](#))

76.121.7 TWordHelper.TryParse

Synopsis: Try to convert a string to a `Word`, report success or failure.

Declaration: `class function TryParse(const AString: string; out AValue: Word)
: Boolean; Static`

Visibility: public

Description: `TryParse` attempts to convert the string `AString` to a `Word`, and reports the success of the attempt. If the attempt is successful, then `True` is returned, and the actual value of the `Word` is returned in `AValue`.

It uses the `val` ([1609](#)) function to perform the conversion, so no localization is taken into account.

See also: `TWordHelper.Parse` ([2001](#)), `Val` ([1570](#))

76.121.8 TWordHelper.ToBoolean

Synopsis: Convert to a boolean value.

Declaration: `function ToBoolean : Boolean`

Visibility: public

Description: `ToBoolean` converts the `Word` value to a boolean: it returns `True` if the value is nonzero, `False` if it is zero.

See also: `TWordHelper.ToSingle` ([2003](#)), `TWordHelper.ToDouble` ([2002](#)), `TWordHelper.ToExtended` ([2003](#)), `TWordHelper.ToString` ([2002](#)), `TWordHelper.ToHexString` ([2003](#))

76.121.9 TWordHelper.ToDouble

Synopsis: Convert to a double-sized floating point value.

Declaration: `function ToDouble : Double`

Visibility: public

Description: `ToDouble` converts the `Word` value to a double-sized floating point value.

See also: `TWordHelper.ToBoolean` ([2002](#)), `TWordHelper.ToExtended` ([2003](#)), `TWordHelper.ToSingle` ([2003](#)), `TWordHelper.ToString` ([2002](#)), `TWordHelper.ToHexString` ([2003](#))

76.121.10 TWordHelper.ToExtended

Synopsis: Convert to an extended-sized floating point value.

Declaration: `function ToExtended : Extended`

Visibility: public

Description: `ToDouble` converts the `Word` value to an extended-sized floating point value.

See also: `TWordHelper.ToBoolean` (2002), `TWordHelper.ToSingle` (2003), `TWordHelper.ToDouble` (2002), `TWordHelper.ToString` (2002), `TWordHelper.ToHexString` (2003)

76.121.11 TWordHelper.ToBinString

Synopsis: Convert to a binary string representation.

Declaration: `function ToBinString : string`

Visibility: public

Description: `ToBinString` converts the `word` value to a binary string representation, showing all 16 bits.

See also: `TWordHelper.ToHexString` (2003), `TWordHelper.ToString` (2002)

76.121.12 TWordHelper.ToHexString

Synopsis: Convert to a hexadecimal string representation.

Declaration: `function ToHexString(const AMinDigits: Integer) : string; Overload`
`function ToHexString : string; Overload`

Visibility: public

Description: `ToHexString` converts the `Word` value to a hexadecimal string representation. The `AMinDigits` argument specifies the minimal number of characters in the resulting string. The string will be left-padded with zeroes if the representation contains less than `AMinDigits` characters.

See also: `TWordHelper.ToBoolean` (2002), `TWordHelper.ToSingle` (2003), `TWordHelper.ToDouble` (2002), `TWordHelper.ToString` (2002), `TWordHelper.ToExtended` (2003)

76.121.13 TWordHelper.ToSingle

Synopsis: Convert to an single-sized floating point value.

Declaration: `function ToSingle : Single`

Visibility: public

Description: `ToSingle` converts the `Word` value to a single-sized floating point value.

See also: `TWordHelper.ToBoolean` (2002), `TWordHelper.ToDouble` (2002), `TWordHelper.ToExtended` (2003), `TWordHelper.ToString` (2002), `TWordHelper.ToHexString` (2003)

76.121.14 TWordHelper.SetBit

Synopsis: Set bit to 1.

Declaration: `function SetBit(const Index: TWordBitIndex) : Word`

Visibility: public

Description: `SetBit` puts value 1 to bit number determined by argument `Index`.

See also: `TWordHelper.ClearBit` (2004), `TWordHelper.ToggleBit` (2004), `TWordHelper.TestBit` (2004), `TWordHelper.HighestSetBitPos` (2005), `TWordHelper.LowestSetBitPos` (2005), `TWordHelper.SetBitsCount` (2005), `TWordHelper.Bits` (2006)

76.121.15 TWordHelper.ClearBit

Synopsis: Set bit to 0.

Declaration: `function ClearBit(const Index: TWordBitIndex) : Word`

Visibility: public

Description: `ClearBit` puts value 0 to bit number determined by argument `Index`.

See also: `TWordHelper.SetBit` (2004), `TWordHelper.ToggleBit` (2004), `TWordHelper.TestBit` (2004), `TWordHelper.HighestSetBitPos` (2005), `TWordHelper.LowestSetBitPos` (2005), `TWordHelper.SetBitsCount` (2005), `TWordHelper.Bits` (2006)

76.121.16 TWordHelper.ToggleBit

Synopsis: Invert bit.

Declaration: `function ToggleBit(const Index: TWordBitIndex) : Word`

Visibility: public

Description: `ToggleBit` reads bit value from bit number determined by argument `Index`, inverts it (if it was 0 it becomes 1, and vice versa), and stores it back.

See also: `TWordHelper.SetBit` (2004), `TWordHelper.ToggleBit` (2004), `TWordHelper.TestBit` (2004), `TWordHelper.HighestSetBitPos` (2005), `TWordHelper.LowestSetBitPos` (2005), `TWordHelper.SetBitsCount` (2005), `TWordHelper.Bits` (2006)

76.121.17 TWordHelper.TestBit

Synopsis: Check bit.

Declaration: `function TestBit(const Index: TWordBitIndex) : Boolean`

Visibility: public

Description: `TestBit` reads boolean value (true if bit is 1, and false if bit is 0) from bit number determined by argument `Index`.

See also: `TWordHelper.SetBit` (2004), `TWordHelper.ToggleBit` (2004), `TWordHelper.TestBit` (2004), `TWordHelper.HighestSetBitPos` (2005), `TWordHelper.LowestSetBitPos` (2005), `TWordHelper.SetBitsCount` (2005), `TWordHelper.Bits` (2006)

76.121.18 TWordHelper.Clear

Synopsis: Set value to 0.

Declaration: `procedure Clear`

Visibility: `public`

76.121.19 TWordHelper.HighestSetBitPos

Synopsis: Return the position of the leftmost bit set.

Declaration: `function HighestSetBitPos : Int8`

Visibility: `public`

Description: `HighestSetBitPos` scans the word, starting at the leftmost position and working towards position 0, and returns the index of the first set bit. The position is measured from the 0-th, rightmost bit. When there are no bits set, the result is -1.

See also: `System.BsrByte` ([1609](#)), `TWordHelper.SetBit` ([2004](#)), `TWordHelper.ClearBit` ([2004](#)), `TWordHelper.ToggleBit` ([2004](#)), `TWordHelper.TestBit` ([2004](#)), `TWordHelper.LowestSetBitPos` ([2005](#)), `TWordHelper.SetBitsCount` ([2005](#)), `TWordHelper.Bits` ([2006](#))

76.121.20 TWordHelper.LowestSetBitPos

Synopsis: Return the position of the rightmost bit set.

Declaration: `function LowestSetBitPos : Int8`

Visibility: `public`

Description: `LowestSetBitPos` scans the word, starting at position 0 (rightmost position) and returns the index of the first set bit. The position is measured from the 0-th, rightmost bit. When there are no bits set, the result is -1.

See also: `System.BsfByte` ([1609](#)), `TWordHelper.SetBit` ([2004](#)), `TWordHelper.ClearBit` ([2004](#)), `TWordHelper.ToggleBit` ([2004](#)), `TWordHelper.TestBit` ([2004](#)), `TWordHelper.HighestSetBitPos` ([2005](#)), `TWordHelper.SetBitsCount` ([2005](#)), `TWordHelper.Bits` ([2006](#))

76.121.21 TWordHelper.SetBitsCount

Synopsis: Count number of bits set.

Declaration: `function SetBitsCount : Byte`

Visibility: `public`

Description: `SetBitsCount` returns the number of bits set in a word.

See also: `System.PopCnt` ([1609](#)), `TWordHelper.SetBit` ([2004](#)), `TWordHelper.ClearBit` ([2004](#)), `TWordHelper.ToggleBit` ([2004](#)), `TWordHelper.TestBit` ([2004](#)), `TWordHelper.HighestSetBitPos` ([2005](#)), `TWordHelper.LowestSetBitPos` ([2005](#)), `TWordHelper.Bits` ([2006](#))

76.121.22 TWordHelper.Bits

Synopsis: Read or write a bit.

Declaration: Property Bits[aIndex: TWordBitIndex]: Boolean

Visibility: public

Access: Read,Write

Description: Bits property reads or writes boolean value to a bit number determined by argument aIndex.

```
MyWord.Clear; // %000000000000000000 MyWord equals 0
MyWord.Bits[0] := true; // %000000000000000001 MyWord equals 1
MyWord.Bits[2] := true; // %00000000000000101 MyWord equals 5
WriteLn(MyWord.Bits[2]); // TRUE
```

See also: TWordHelper.SetBit ([2004](#)), TWordHelper.ClearBit ([2004](#)), TWordHelper.ToggleBit ([2004](#)), TWordHelper.TestBit ([2004](#)), TWordHelper.HighestSetBitPos ([2005](#)), TWordHelper.LowestSetBitPos ([2005](#)), TWordHelper.SetBitsCount ([2005](#)), TWordHelper.Nibbles ([2006](#)), TWordHelper.Bytes ([2006](#))

76.121.23 TWordHelper.Nibbles

Synopsis: Read or write a nibble.

Declaration: Property Nibbles[aIndex: TWordNibbleIndex]: nibble

Visibility: public

Access: Read,Write

Description: Nibbles property reads or writes nibble value to a nibble number determined by argument aIndex.

```
MyWord.Clear; // %000000000000000000 MyWord equals 0
MyWord.Nibbles[1] := 3; // %000000000000110000 MyWord equals 48
WriteLn(MyWord.Nibbles[1]); // 3
```

See also: TWordHelper.Bits ([2006](#)), TWordHelper.Bytes ([2006](#))

76.121.24 TWordHelper.Bytes

Synopsis: Read or write a byte.

Declaration: Property Bytes[aIndex: TWordByteIndex]: Byte

Visibility: public

Access: Read,Write

Description: Bytes property reads or writes byte value to a byte number determined by argument aIndex.

```
MyWord.Clear; // %000000000000000000 MyWord equals 0
MyWord.Bytes[1] := 1; // %000000001000000000 MyWord equals 256
WriteLn(MyWord.Bytes[1]); // 1
```

See also: TWordHelper.Bits ([2006](#)), TWordHelper.Nibbles ([2006](#))

Chapter 77

Reference for unit 'Types'

77.1 Used units

Table 77.1: Used units by unit 'Types'

Name	Page
System	1335

77.2 Overview

Starting with D6, types from Windows specific units that were needed in Kylix were extracted to this unit. So it mostly contains type of Windows origin that are needed in the VCL framework.

77.3 Constants, types and variables

77.3.1 Constants

`cPI : Single = 3.141592654`

`cPIdiv180 : Single = 0.017453292`

`cPIdiv2 : Single = 1.570796326`

`cPIdiv4 : Single = 0.785398163`

`CurveKappa = 0.5522847498`

`CurveKappaInv = 1 - CurveKappa`

Epsilon : Single = 1E-40

Epsilon2 : Single = 1E-30

E_FAIL = HRESULT(\$80004005)

Defined for Delphi compatibility, this should not be used.

E_INVALIDARG = HRESULT(\$80070057)

Defined for Delphi compatibility, this should not be used.

GUID_NULL : TGuid = '{00000000-0000-0000-0000-000000000000}'

GUID_NULL is the definition of the NULL (empty) GUID.

LOCK_EXCLUSIVE = 2

Defined for Delphi compatibility, this should not be used.

LOCK_ONLYONCE = 4

Defined for Delphi compatibility, this should not be used.

LOCK_WRITE = 1

Defined for Delphi compatibility, this should not be used.

STATFLAG_DEFAULT = 0

Defined for Delphi compatibility, this should not be used.

STATFLAG_NONAME = 1

Defined for Delphi compatibility, this should not be used.

STATFLAG_NOOPEN = 2

Defined for Delphi compatibility, this should not be used.

STGTY_LOCKBYTES = 3

Defined for Delphi compatibility, this should not be used.

STGTY_PROPERTY = 4

Defined for Delphi compatibility, this should not be used.

STGTY_STORAGE = 1

Defined for Delphi compatibility, this should not be used.

STGTY_STREAM = 2

Defined for Delphi compatibility, this should not be used.

STG_E_ABNORMALAPIEXIT = HRESULT(\$800300FA)

Defined for Delphi compatibility, this should not be used.

STG_E_ACCESSDENIED = HRESULT(\$80030005)

Defined for Delphi compatibility, this should not be used.

STG_E_BADBASEADDRESS = HRESULT(\$80030110)

Defined for Delphi compatibility, this should not be used.

STG_E_CANTSAVE = HRESULT(\$80030103)

Defined for Delphi compatibility, this should not be used.

STG_E_DISKISWRITEPROTECTED = HRESULT(\$80030013)

Defined for Delphi compatibility, this should not be used.

STG_E_DOCFILECORRUPT = HRESULT(\$80030109)

Defined for Delphi compatibility, this should not be used.

STG_E_EXTANTMARSHALLINGS = HRESULT(\$80030108)

Defined for Delphi compatibility, this should not be used.

STG_E_FILEALREADYEXISTS = HRESULT(\$80030050)

Defined for Delphi compatibility, this should not be used.

STG_E_FILENOTFOUND = HRESULT(\$80030002)

Defined for Delphi compatibility, this should not be used.

STG_E_INCOMPLETE = HRESULT(\$80030201)

Defined for Delphi compatibility, this should not be used.

STG_E_INSUFFICIENTMEMORY = HRESULT(\$80030008)

Defined for Delphi compatibility, this should not be used.

STG_E_INUSE = HRESULT(\$80030100)

Defined for Delphi compatibility, this should not be used.

STG_E_INVALIDFLAG = HRESULT(\$800300FF)

Defined for Delphi compatibility, this should not be used.

STG_E_INVALIDFUNCTION = HRESULT(\$80030001)

Defined for Delphi compatibility, this should not be used.

STG_E_INVALIDHANDLE = HRESULT(\$80030006)

Defined for Delphi compatibility, this should not be used.

STG_E_INVALIDHEADER = HRESULT(\$800300FB)

Defined for Delphi compatibility, this should not be used.

STG_E_INVALIDNAME = HRESULT(\$800300FC)

Defined for Delphi compatibility, this should not be used.

STG_E_INVALIDPARAMETER = HRESULT(\$80030057)

Defined for Delphi compatibility, this should not be used.

STG_E_INVALIDPOINTER = HRESULT(\$80030009)

Defined for Delphi compatibility, this should not be used.

STG_E_LOCKVIOLATION = HRESULT(\$80030021)

Defined for Delphi compatibility, this should not be used.

STG_E_MEDIUMFULL = HRESULT(\$80030070)

Defined for Delphi compatibility, this should not be used.

STG_E_NOMOREFILES = HRESULT(\$80030012)

Defined for Delphi compatibility, this should not be used.

STG_E_NOTCURRENT = HRESULT(\$80030101)

Defined for Delphi compatibility, this should not be used.

STG_E_OLDDLL = HRESULT(\$80030105)

Defined for Delphi compatibility, this should not be used.

STG_E_OLDFORMAT = HRESULT(\$80030104)

Defined for Delphi compatibility, this should not be used.

STG_E_PATHNOTFOUND = HRESULT(\$80030003)

Defined for Delphi compatibility, this should not be used.

STG_E_PROPSETMISMATCHED = HRESULT(\$800300F0)

Defined for Delphi compatibility, this should not be used.

STG_E_READFAULT = HRESULT(\$8003001E)

Defined for Delphi compatibility, this should not be used.

STG_E_REVERTED = HRESULT(\$80030102)

Defined for Delphi compatibility, this should not be used.

STG_E_SEEKERROR = HRESULT(\$80030019)

Defined for Delphi compatibility, this should not be used.

STG_E_SHAREREQUIRED = HRESULT(\$80030106)

Defined for Delphi compatibility, this should not be used.

STG_E_SHAREVIOLATION = HRESULT(\$80030020)

Defined for Delphi compatibility, this should not be used.

STG_E_TERMINATED = HRESULT(\$80030202)

Defined for Delphi compatibility, this should not be used.

STG_E_TOOMANYOPENFILES = HRESULT(\$80030004)

Defined for Delphi compatibility, this should not be used.

STG_E_UNIMPLEMENTEDFUNCTION = HRESULT(\$800300FE)

Defined for Delphi compatibility, this should not be used.

STG_E_UNKNOWN = HRESULT(\$800300FD)

Defined for Delphi compatibility, this should not be used.

STG_E_WRITEFAULT = HRESULT(\$8003001D)

Defined for Delphi compatibility, this should not be used.

STG_S_BLOCK = \$00030201

Defined for Delphi compatibility, this should not be used.

```
STG_S_CONVERTED = $00030200
```

Defined for Delphi compatibility, this should not be used.

```
STG_S_MONITORING = $00030203
```

Defined for Delphi compatibility, this should not be used.

```
STG_S_RETRYNOW = $00030202
```

Defined for Delphi compatibility, this should not be used.

```
STREAM_SEEK_CUR = 1
```

Defined for Delphi compatibility, this should not be used.

```
STREAM_SEEK_END = 2
```

Defined for Delphi compatibility, this should not be used.

```
STREAM_SEEK_SET = 0
```

Defined for Delphi compatibility, this should not be used.

77.3.2 Types

```
ArgList = Pointer
```

ArgList is defined for Delphi/Kylix compatibility and should not be used.

```
DWORD = LongWord
```

Alias for cardinal type.

```
FILETIME = _FILETIME
```

Alias for the `_FILETIME` type.

```
Largeint = Int64
```

Largeint is an alias for the `Int64` type defined in the system unit. This is an alias for Delphi/Kylix compatibility.

```
LargeUInt = QWord
```

LargeUInt is an alias for the `QWord` type defined in the system unit. This is an alias for Delphi/Kylix compatibility.

```
LARGE_INT = Largeint
```

LARGE_INT is an alias for the Int64 type defined in the system unit. This is an alias for Delphi/Kylix compatibility.

LARGE_UINT = LargeUInt

LARGE_UINT is an alias for the QWord type defined in the system unit. This is an alias for Delphi/Kylix compatibility.

PByte = System.PByte

PByte is defined in the system unit. This is an alias for Delphi/Kylix compatibility.

PCLSID = PGuid

PCLSID is a pointer to a TCLSID type.

PDisplay = Pointer

PDisplay is defined for Delphi/Kylix compatibility and should not be used.

PDouble = System.PDouble

PDouble is defined in the system unit. This is an alias for Delphi/Kylix compatibility.

PDWord = ^DWORD

PDWord is equivalent to the PCardinal type.

PEvent = Pointer

PEvent is defined for Delphi/Kylix compatibility and should not be used.

PFileTime = ^TFileTime

Pointer to TFileTime type.

PLargeInt = ^LargeInt

PLargeInt is an alias for the PInt64 type defined in the system unit. This is an alias for Delphi/Kylix compatibility.

PLargeUInt = ^LargeUInt

PLargeUInt is an alias for the PQWord type defined in the system unit. This is an alias for Delphi/Kylix compatibility.

PLongint = System.PLongint

PLongint is defined in the system unit. This is an alias for Delphi/Kylix compatibility.

POleStr = PWideChar

`POleStr` is a pointer to a (double) null-terminated array of `TOleChar` characters.

`PPoint` = `^TPoint`

`PPoint` is a typed pointer to the `TPoint` (2026) type.

`PPointF` = `^TPointF`

`PPoleStr` = `^POleStr`

`PPoleStr` is a typed pointer to a `POleStr` variable.

`PRect` = `^TRect`

`PRect` is a typed pointer to the `TRect` (2037) type.

`PRectF` = `^TRectF`

`PSize` = `^TSize`

`PSize` is a typed pointer to the `TSize` (2047) type.

`PSizeF` = `^TSizeF`

`PSmallInt` = `System.PSmallInt`

`PSmallInt` is defined in the system unit. This is an alias for Delphi/Kylix compatibility.

`PSmallPoint` = `^TSmallPoint`

`PSmallPoint` is a typed pointer to the `TSmallPoint` (2018) record.

`PStatStg` = `^TStatStg`

Pointer to `TStatStg` record.

`PXrmOptionDescRec` = `^TXrmOptionDescRec`

`PXrmOptionDescRec` is defined for Delphi/Kylix compatibility and should not be used.

`Region` = `Pointer`

`Region` is defined for Delphi/Kylix compatibility and should not be used.

`STATSTG` = `TStatStg`

Alias for the `TStatStg` type.

`TagPoint` = `TPoint` deprecated

tagPOINT is a simple alias for TPoint (2026)

TagSize = TSize deprecated

tagSize is an alias for the TSize (2047) type.

```
tagSTATSTG = record
public
  pwcsName : POleStr;
  dwType : DWORD
;
  cbSize : LARGE_UINT;
  mtime : TFileTime;
  ctime : TFileTime
;
  atime : TFileTime;
  grfMode : DWORD;
  grfLocksSupported : DWORD
;
  clsid : TCLSID;
  grfStateBits : DWORD;
  reserved : DWORD;
end
```

tagSTATSTG is used in the IStream.Stat (2057) call. It describes a storage medium (typically a file).

TAnsiStringDynArray = Array of AnsiString

TArray4IntegerType = Array[0..3] of LongInt

TBooleanDynArray = Array of Boolean

TBooleanDynArray is a standard definition of a dynamical array of booleans.

TByteDynArray = Array of Byte

TByteDynArray is a standard definition of a dynamical array of (8-bit, unsigned) bytes.

TCardinalDynArray = Array of Cardinal

TCardinalDynArray is a standard definition of a dynamical array of (32-bit, unsigned) cardinals.

TClassicByteDynArray = TByteDynArray

TClassicStringDynArray = TStringDynArray

TCLSID = TGuid

TCLSID is an alias for the #rtl.system.TGUID (1583) type.

TCompDynArray = Array of Comp

TCurrencyArray = Array of currency

TDirection = (FromBeginning, FromEnd)

Table 77.2: Enumeration values for type TDirection

Value	Explanation
FromBeginning	
FromEnd	

TDoubleDynArray = Array of Double

TSoubleDynArray is a standard definition of a dynamical array of doubles. (regular floating point type)

TDuplicates = (dupIgnore, dupAccept, dupError)

Table 77.3: Enumeration values for type TDuplicates

Value	Explanation
dupAccept	Accept duplicates, adding them to the list.
dupError	Raise an error when an attempt is made to add a duplicate.
dupIgnore	Ignore the new item, do not add it to the list.

TDuplicates can be used to indicate how a list structure acts on the addition of a duplicate item to the list.

dupIgnore Ignore the new item, do not add it to the list.

dupAccept Accept duplicates, adding them to the list.

dupError Raise an error when an attempt is made to add a duplicate.

TEndian = Objpas.TEndian

TExtendedDynArray = Array of Extended

TFileTime = _FILETIME

Alias for the _FILETIME type.

`THorzRectAlign = (Center, Left, Right)`

Table 77.4: Enumeration values for type `THorzRectAlign`

Value	Explanation
Center	
Left	
Right	

`TInt64DynArray = Array of Int64`

`TInt64DynArray` is a standard definition of a dynamical array of (64-bit, signed) `int64s`.

`TIntegerDynArray = Array of Integer`

`TIntegerDynArray` is a standard definition of a dynamical array of (32-bit, signed) integers.

`TListCallback = procedure(data: pointer; arg: pointer) of object`

`TListCallback` is the prototype for a `Foreach` operation on a list. It will be called with as `Data` the pointer in the list, and `Arg` will contain the extra user data added to the `Foreach` call. It can be used in methods of objects; for a version that can be used as a global procedure, see `TListStaticCallback` (2017)

`TListStaticCallback = procedure(data: pointer; arg: pointer)`

`TListStaticCallback` is the prototype for a `Foreach` operation on a list. It will be called with as `Data` the pointer in the list, and `Arg` will contain the extra user data added to the `Foreach` call. It can be used in plain procedures; for a version that can be used as a method, see `TListCallback` (2017)

`TLongWordDynArray = Array of LongWord`

`TLongWordDynArray` is a standard definition of a dynamical array of (32-bit, unsigned) `LongWords`.

`TObjectDynArray = Array of TObject`

`ToleChar = WideChar`

`ToleChar` is an alias for the `WideChar` type, defined in the system unit.

`TPointerDynArray = Array of Pointer`

Dynamic array of untyped pointers.

`TQWordDynArray = Array of QWord`

`TQWordDynArray` is a standard definition of a dynamical array of (64-bit, unsigned) `QWords`.

TRTLStringDynArray = Array of Ansistring

TShortIntDynArray = Array of ShortInt

TShortintDynArray is a standard definition of a dynamical array of (8-bit, signed) shortints.

TSingleDynArray = Array of Single

TSingleDynArray is a standard definition of a dynamical array of singles. (smallest floating point type)

TSmallIntDynArray = Array of SmallInt

TSmallintDynArray is a standard definition of a dynamical array of (16-bit, unsigned) integers.

```
TSmallPoint = packed record
public
  X : SmallInt;
  Y : SmallInt
;
end
```

TSmallPoint defines a point in a 2-dimensional plane, just like TPoint (2026), but the coordinates have a smaller range: The coordinates are smallints (16-bit, signed) and they run from -MaxSmallInt to maxSmallint.

TSplitRectType = (srLeft, srRight, srTop, srBottom)

Table 77.5: Enumeration values for type TSplitRectType

Value	Explanation
srBottom	
srLeft	
srRight	
srTop	

TStatStg = tagSTATSTG

TStatStg is a record type describing a storage medium. It is uses in the IStream.Stat (2057) function.

TStringDynArray = Array of AnsiString

TStringDynArray is a standard definition of a dynamical array of Ansistrings.

TUnicodeStringDynArray = Array of UnicodeString

TValueRelationship = - 1..1

TVertRectAlign = (Center, Top, Bottom)

Table 77.6: Enumeration values for type TVertRectAlign

Value	Explanation
Bottom	
Center	
Top	

TWideStringDynArray = Array of WideString

TWideStringDynArray is a standard definition of a dynamical array of WideStrings.

TWordDynArray = Array of Word

TWordDynArray is a standard definition of a dynamical array of (16-bit, unsigned) words.

TXrmOptionDescRec = record
end

TXrmOptionDescRec is defined for Delphi/Kylix compatibility and should not be used.

Widget = Pointer

Widget is defined for Delphi/Kylix compatibility and should not be used.

WidgetClass = Pointer

WidgetClass is defined for Delphi/Kylix compatibility and should not be used.

XrmOptionDescRec = TXrmOptionDescRec

XrmOptionDescRec is defined for Delphi/Kylix compatibility and should not be used.

```
_FILETIME = packed record
public
  dwLowDateTime : DWORD;
  dwHighDateTime
  : DWORD;
end
```

_FILETIME describes a file time stamp. It is defined for Delphi/Kylix compatibility and should not be used except when implementing or accessing the IStream interface. The TDateTime type should be used instead.

77.4 Procedures and functions

77.4.1 Bounds

Synopsis: Create a rectangle, given a position and size.

Declaration: `function Bounds(ALeft: Integer; ATop: Integer; AWidth: Integer; AHeight: Integer) : TRect`

Visibility: default

Description: `Bounds` returns a `TRect` structure with the indicated position (`Left=ALeft` and `Top=ATop`) and size (`Right=ALeft+AWidth` and `Bottom=ATop+AHeight`)

See also: `Rect` ([2023](#)), `PtInRect` ([2023](#)), `IntersectRect` ([2021](#)), `UnionRect` ([2024](#))

77.4.2 CenteredRect

Declaration: `function CenteredRect(const SourceRect: TRect; const aCenteredRect: TRect) : TRect`

Visibility: default

77.4.3 CenterPoint

Synopsis: Return the center point of a rectangle.

Declaration: `function CenterPoint(const Rect: TRect) : TPoint`

Visibility: default

Description: `CenterPoint` returns the center point of the rectangle `Rect`.

See also: `PtInRect` ([2023](#)), `IntersectRect` ([2021](#)), `IsRectEmpty` ([2021](#)), `OffsetRect` ([2022](#)), `InflateRect` ([2021](#)), `Size` ([2024](#)), `IsRectEmpty` ([2021](#))

77.4.4 EqualRect

Synopsis: Check if two rectangles are equal.

Declaration: `function EqualRect(const r1: TRect; const r2: TRect) : Boolean`
`function EqualRect(const r1: TRectF; const r2: TRectF) : Boolean`

Visibility: default

Description: `EqualRect` returns `True` if the rectangles `R1` and `R2` are equal (i.e. have the position and size). If the rectangles differ, the function returns `False`

See also: `Rect` ([2023](#)), `Bounds` ([2020](#)), `PtInRect` ([2023](#)), `IntersectRect` ([2021](#)), `UnionRect` ([2024](#)), `IsRectEmpty` ([2021](#)), `OffsetRect` ([2022](#)), `InflateRect` ([2021](#)), `Size` ([2024](#))

77.4.5 InflateRect

Synopsis: Increase the rectangle in size, keeping it centered.

Declaration: `function InflateRect (var Rect: TRect; dx: Integer; dy: Integer) : Boolean`
`function InflateRect (var Rect: TRectF; dx: single; dy: Single) : Boolean`

Visibility: default

Description: `InflateRect` inflates the rectangle horizontally with `dx` pixels on each side, and vertically with `dy` pixels, thus keeping its center point on the same location. It returns `true` if the operation was successfully, `False` if it was not (only possible if the address of `Rect` is `Nil`).

See also: `PtinRect` (2023), `IntersectRect` (2021), `IsRectEmpty` (2021), `OffsetRect` (2022), `CenterPoint` (2020), `Size` (2024), `IsRectEmpty` (2021)

77.4.6 IntersectRect

Synopsis: Return the intersection of 2 rectangles.

Declaration: `function IntersectRect (const Rect1: TRect; const Rect2: TRect) : Boolean`
`function IntersectRect (const Rect1: TRectF; const Rect2: TRectF) : Boolean`
`function IntersectRect (var Rect: TRect; const R1: TRect; const R2: TRect) : Boolean`
`function IntersectRect (var Rect: TRectF; const R1: TRectF; const R2: TRectF) : Boolean`

Visibility: default

Description: `IntersectRect` returns the intersection of the 2 rectangles `R1` and `R2` in `Rect`. It returns `True` if the 2 rectangles have an intersection, otherwise `False` is returned, and `Rect` is filled with zero.

See also: `PtinRect` (2023), `UnionRect` (2024), `IsRectEmpty` (2021), `OffsetRect` (2022), `InflateRect` (2021), `Size` (2024)

77.4.7 IntersectRectF

Declaration: `function IntersectRectF (out Rect: TRectF; const R1: TRectF; const R2: TRectF) : Boolean`

Visibility: default

77.4.8 IsRectEmpty

Synopsis: Check whether a rectangle is empty.

Declaration: `function IsRectEmpty (const Rect: TRectF) : Boolean`
`function IsRectEmpty (const Rect: TRect) : Boolean`

Visibility: default

Description: `IsRectEmpty` returns `true` if the rectangle is empty, i.e. has a zero or negative width or height.

See also: `PtinRect` (2023), `IntersectRect` (2021), `IsRectEmpty` (2021), `OffsetRect` (2022), `InflateRect` (2021), `Size` (2024)

77.4.9 MinPoint

```

Declaration: function MinPoint(const P1: TPointF; const P2: TPointF) : TPointF
              ; Overload
              function MinPoint(const P1: TPoint; const P2: TPoint) : TPoint
              ; Overload

```

Visibility: default

77.4.10 MultiplyRect

```
Declaration: procedure MultiplyRect(var R: TRectF; const DX: Single;
                                   const DY: Single)
```

Visibility: default

77.4.11 NormalizeRect

Declaration: `function NormalizeRect(const ARect: TRectF) : TRectF; Overload`

Visibility: default

77.4.12 NormalizeRectF

Declaration: function NormalizeRectF(const Pts: Array of TPointF) : TRectF; Overload

Visibility: default

77.4.13 OffsetRect

Synopsis: Offset the rectangle.

```
Declaration: function OffsetRect(var Rect: TRect; DX: Integer; DY: Integer) : Boolean
            function OffsetRect(var Rect: TRectF; DX: Single; DY: Single) : Boolean
```

Visibility: default

Description: `OffsetRect` offsets the rectangle `Rect` by a horizontal distance `DX` and a vertical distance `DY`. The operation returns `True` if the operation was successful, `false` if it was not (only possible if the address of `Rect` is `Nil`).

See also: [PtinRect \(2023\)](#), [IntersectRect \(2021\)](#), [IsRectEmpty \(2021\)](#), [OffsetRect \(2022\)](#), [InflateRect \(2021\)](#), [Size \(2024\)](#), [IsRectEmpty \(2021\)](#)

77.4.14 Point

Synopsis: Create a point.

```
Declaration: function Point(x: Integer; y: Integer) : TPoint
```

Visibility: default

Description: Point returns a TPoint structure with the given position (X, Y).

See also: Rect ([2023](#)), PtInRect ([2023](#))

77.4.15 PointF

Declaration: `function PointF(x: Single; y: Single) : TPointF`

Visibility: default

77.4.16 PtInRect

Synopsis: Check whether a point is inside a rectangle.

Declaration: `function PtInRect(const Rect: TRect; const p: TPoint) : Boolean`
`function PtInRect(const Rect: TRectF; const p: TPointF) : Boolean`

Visibility: default

Description: `PtInRect` returns `True` if `p` is located inside `Rect`, and `False` if it is located outside the rectangle.

Remark Note that the bottom, right edges are not considered part of the rectangle, therefore a point located on one of these edges will not be considered part of the rectangle, meaning that for a record (10,10,100,100) the point (90,100) will not be considered part of the record, but 90, 10 will be.

See also: `IntersectRect` (2021), `UnionRect` (2024), `IsRectEmpty` (2021), `OffsetRect` (2022), `InflateRect` (2021), `Size` (2024)

77.4.17 Rect

Synopsis: Create a rectangle record.

Declaration: `function Rect(Left: Integer; Top: Integer; Right: Integer;
Bottom: Integer) : TRect`

Visibility: default

Description: `Rect` returns a rectangle structure with the 4 members `Left`, `Top`, `Right` and `Bottom` as passed in the arguments.

See also: `Bounds` (2020), `PtInRect` (2023), `IntersectRect` (2021), `UnionRect` (2024), `IsRectEmpty` (2021), `OffsetRect` (2022), `InflateRect` (2021), `Size` (2024)

77.4.18 RectCenter

Declaration: `function RectCenter(var R: TRect; const Bounds: TRect) : TRect`
`function RectCenter(var R: TRectF; const Bounds: TRectF) : TRectF`

Visibility: default

77.4.19 RectF

Declaration: `function RectF(Left: Single; Top: Single; Right: Single; Bottom: Single)
: TRectF`

Visibility: default

77.4.20 RectHeight

```
Declaration: function RectHeight(const Rect: TRect) : Integer
              function RectHeight(const Rect: TRectF) : Single
```

Visibility: default

77.4.21 RectWidth

```
Declaration: function RectWidth(const Rect: TRect) : Integer
             function RectWidth(const Rect: TRectF) : Single
```

Visibility: default

77.4.22 ScalePoint

```

Declaration: function ScalePoint(const P: TPointF; dX: Single; dY: Single) : TPointF
                ; Overload
  function ScalePoint(const P: TPoint; dX: Single; dY: Single) : TPoint
                ; Overload

```

Visibility: default

77.4.23 Size

Synopsis: Return the size of the rectangle.

```
Declaration: function Size(AWidth: Integer; AHeight: Integer) : TSize
             function Size(const ARect: TRect) : TSize
```

Visibility: default

Description: Size returns a TSize record with the indicated AWidth, AHeight. In the case ARect is passed, the width and height are calculated (taking into account that the right, bottom are not considered part of the rectangle).

See also: [PtInRect \(2023\)](#), [IntersectRect \(2021\)](#), [IsRectEmpty \(2021\)](#), [OffsetRect \(2022\)](#), [InflateRect \(2021\)](#), [CenterPoint \(2020\)](#), [IsRectEmpty \(2021\)](#)

77.4.24 SplitRect

```
Declaration: function SplitRect(const Rect: TRect; SplitType: TSplitRectType;
                               Size: Integer) : TRect; Overload
function SplitRect(const Rect: TRect; SplitType: TSplitRectType;
                  Percent: Double) : TRect; Overload
```

Visibility: default

77.4.25 UnionRect

Synopsis: Return the union of 2 rectangles.

Declaration: `function UnionRect (var Rect: TRect; const R1: TRect; const R2: TRect) : Boolean`
`function UnionRect (var Rect: TRectF; const R1: TRectF; const R2: TRectF) : Boolean`
`function UnionRect (const R1: TRect; const R2: TRect) : TRect`
`function UnionRect (const R1: TRectF; const R2: TRectF) : TRectF`

Visibility: default

Description: `UnionRect` returns the rectangle that encompasses both `R1` and `R2` in `Rect`. It returns `True` if the resulting rectangle is not empty, `False` if the result is an empty rectangle (in which case the result is filled with zeroes)

See also: `PtinRect` ([2023](#)), `IntersectRect` ([2021](#)), `IsRectEmpty` ([2021](#)), `OffsetRect` ([2022](#)), `InflateRect` ([2021](#)), `Size` ([2024](#))

77.4.26 UnionRectF

Declaration: `function UnionRectF (out Rect: TRectF; const R1: TRectF; const R2: TRectF) : Boolean`

Visibility: default

77.5 TPoint

`TPoint` = packed record
public
`X : LongInt;`
`Y : LongInt;`
class
`function Zero : TPoint; Static;`
`function Add (const apt: TPoint ) : TPoint;`
`function Distance (const apt: TPoint) : ValReal;`
`function IsZero : Boolean;`
`function Subtract (const apt: TPoint) : TPoint`
`;`
`SetLocation;`
`Offset;`
`function Angle (const pt: TPoint) : Single`
`;`
class `function PointInCircle (const apt: TPoint; const acenter : TPoint;`
`const aradius: Integer)`
`: Boolean; Static;`
`TPoint.class operator =(const apt1: TPoint;`
`const apt2: TPoint)`
`: Boolean;`
`TPoint`
`.class operator <> (const apt1: TPoint; const apt2: TPoint)`
`: Boolean;`
`TPoint.class operator + (const apt1`

```

: TPoint; const apt2: TPoint) : TPoint;
TPoint.class operator -
(const apt1: TPoint; const apt2: TPoint) : TPoint;
TPoint.class
operator :=(const aspt: TSmallPoint) : TPoint;
TPoint.class operator
explicit(const apt: TPoint) : TSmallPoint;
end

```

`TPoint` is a generic definition of a point in a 2-dimensional discrete plane, where `X` indicates the horizontal position, and `Y` the vertical position (positions usually measured in pixels), and `0, 0` is the origin of the plane.

Usually, the origin is the upper-left corner of the screen, with `Y` increasing as one moves further down the screen - this is opposite to the mathematical view where `Y` increases as one moves upwards.

The coordinates are integers, (32-bit, signed) so the coordinate system runs from `-MaxInt` to `MaxInt`.

77.5.1 Method overview

Page	Method	Description
2026	<code>Add</code>	
2028	<code>add(TPoint,TPoint):TPoint</code>	
2027	<code>Angle</code>	
2028	<code>assign(TSmallPoint):TPoint</code>	
2026	<code>Distance</code>	
2027	<code>equal(TPoint,TPoint):Boolean</code>	
2028	<code>explicit(TPoint):TSmallPoint</code>	
2027	<code>IsZero</code>	
2027	<code>notequal(TPoint,TPoint):Boolean</code>	
2027	<code>Offset</code>	
2027	<code>PointInCircle</code>	
2027	<code>SetLocation</code>	
2027	<code>Subtract</code>	
2028	<code>subtract(TPoint,TPoint):TPoint</code>	
2026	<code>Zero</code>	

77.5.2 TPoint.Zero

Declaration: `class function Zero : TPoint; Static`

Visibility: `public`

77.5.3 TPoint.Add

Declaration: `function Add(const apt: TPoint) : TPoint`

Visibility: `public`

77.5.4 TPoint.Distance

Declaration: `function Distance(const apt: TPoint) : ValReal`

Visibility: `public`

77.5.5 TPoint.IsZero

Declaration: function IsZero : Boolean

Visibility: public

77.5.6 TPoint.Subtract

Declaration: function Subtract(const apt: TPoint) : TPoint

Visibility: public

77.5.7 TPoint.SetLocation

Declaration: procedure SetLocation(const apt: TPoint)
 procedure SetLocation(ax: LongInt; ay: LongInt)

Visibility: public

77.5.8 TPoint.Offset

Declaration: procedure Offset(const apt: TPoint)
 procedure Offset(dx: LongInt; dy: LongInt)

Visibility: public

77.5.9 TPoint.Angle

Declaration: function Angle(const pt: TPoint) : Single

Visibility: public

77.5.10 TPoint.PointInCircle

Declaration: class function PointInCircle(const apt: TPoint; const acenter: TPoint;
 const aradius: Integer) : Boolean; Static

Visibility: public

77.5.11 TPoint.equal(TPoint,TPoint):Boolean

Declaration: TPoint.class operator =(const apt1: TPoint; const apt2: TPoint)
 : Boolean

Visibility: public

77.5.12 TPoint.notequal(TPoint,TPoint):Boolean

Declaration: TPoint.class operator <>(const apt1: TPoint; const apt2: TPoint)
 : Boolean

Visibility: public

77.5.13 TPoint.add(TPoint,TPoint):TPoint

Declaration: `TPoint.class operator +(const apt1: TPoint; const apt2: TPoint) : TPoint`

Visibility: public

77.5.14 TPoint.subtract(TPoint,TPoint):TPoint

Declaration: `TPoint.class operator -(const apt1: TPoint; const apt2: TPoint) : TPoint`

Visibility: public

77.5.15 TPoint.assign(TSmallPoint):TPoint

Declaration: `TPoint.class operator :=(const aspt: TSmallPoint) : TPoint`

Visibility: public

77.5.16 TPoint.explicit(TPoint):TSmallPoint

Declaration: `TPoint.class operator explicit(const apt: TPoint) : TSmallPoint`

Visibility: public

77.6 TPoint3D

```
TPoint3D = packed record
public
  TSingle3Array = Array[0..2] of single
  ;
  constructor Create(const ax: single; const ay: single; const az
    : single);
  Offset;
case Integer of
0: (
public
  data : TSingle3Array
  ;
);
1: (
public
  x : single;
  y : single;
  z : single;
);
end
```

77.6.1 Method overview

Page	Method	Description
2029	Create	
2029	Offset	

77.6.2 TPoint3D.Create

Declaration: constructor Create(const ax: single; const ay: single; const az: single)

Visibility: public

77.6.3 TPoint3D.Offset

Declaration: procedure Offset(const adeltax: single; const adeltay: single;
const adeltaz: single)
procedure Offset(const adelta: TPoint3D)

Visibility: public

77.7 TPointF

```
TPointF = packed record
public
  x : Single;
  y : Single;
  Add;
  function Distance(const apt: TPointF) : Single;
  function DotProduct
    (const apt: TPointF) : Single;
  function IsZero : Boolean;
  Subtract
;
  SetLocation;
  Offset;
  EqualsTo;
  function Scale(afactor: Single
  ) : TPointF;
  function Ceiling : TPoint;
  function Truncate : TPoint
;
  function Floor : TPoint;
  function Round : TPoint;
  function
    &Length : Single;
  function Rotate(angle: single) : TPointF;
  function
    Reflect(const normal: TPointF) : TPointF;
  function MidPoint(const
    b: TPointF) : TPointF;
  PointInCircle;
  class function Zero : TPointF
;  Static;
  function Angle(const b: TPointF) : Single;
  function
    AngleCosine(const b: TPointF) : single;
  function CrossProduct(const
    apt: TPointF) : Single;
  function Normalize : TPointF;
```

```

Create
;
TPointF.class operator =(const apt1: TPointF; const apt2: TPointF
)
                                : Boolean;
TPointF.class operator
<>(const apt1: TPointF; const apt2: TPointF)
: Boolean;
TPointF.class operator +(const apt1: TPointF
; const apt2: TPointF)
                                : TPointF;
TPointF
.class operator -(const apt1: TPointF; const apt2: TPointF)
: TPointF;
TPointF.class operator -(const
apt1: TPointF) : TPointF;
TPointF.class operator *(const apt1:
TPointF; const apt2: TPointF)
                                : TPointF
;
TPointF.class operator *(const apt1: TPointF; afactor: single
) : TPointF;
TPointF.class operator *(afactor: single; const apt1
: TPointF) : TPointF;
TPointF.class operator /(const apt1: TPointF
; afactor: single) : TPointF;
TPointF.class operator :=(const apt
: TPoint) : TPointF;
TPointF.class operator ** (const apt1: TPointF
; const apt2: TPointF)
                                : Single;
end

```

77.7.1 Method overview

Page	Method	Description
2031	Add	
2035	add(TPointF,TPointF):TPointF	
2034	Angle	
2034	AngleCosine	
2035	assign(TPoint):TPointF	
2032	Ceiling	
2034	Create	
2034	CrossProduct	
2031	Distance	
2035	divide(TPointF,single):TPointF	
2032	DotProduct	
2034	equal(TPointF,TPointF):Boolean	
2032	EqualsTo	
2033	Floor	
2032	IsZero	
2033	Length	
2033	MidPoint	
2035	multiply(single,TPointF):TPointF	
2035	multiply(TPointF,single):TPointF	
2035	multiply(TPointF,TPointF):TPointF	
2035	negative(TPointF):TPointF	
2034	Normalize	
2034	notequal(TPointF,TPointF):Boolean	
2032	Offset	
2033	PointInCircle	
2036	power(TPointF,TPointF):Single	
2033	Reflect	
2033	Rotate	
2033	Round	
2032	Scale	
2032	SetLocation	
2032	Subtract	
2035	subtract(TPointF,TPointF):TPointF	
2033	Truncate	
2034	Zero	

77.7.2 TPointF.Add

Declaration: `function Add(const apt: TPoint) : TPointF`
`function Add(const apt: TPointF) : TPointF`

Visibility: public

77.7.3 TPointF.Distance

Declaration: `function Distance(const apt: TPointF) : Single`

Visibility: public

77.7.12 TPointF.Truncate

Declaration: `function Truncate : TPoint`

Visibility: `public`

77.7.13 TPointF.Floor

Declaration: `function Floor : TPoint`

Visibility: `public`

77.7.14 TPointF.Round

Declaration: `function Round : TPoint`

Visibility: `public`

77.7.15 TPointF.Length

Declaration: `function &Length : Single`

Visibility: `public`

77.7.16 TPointF.Rotate

Declaration: `function Rotate(angle: single) : TPointF`

Visibility: `public`

77.7.17 TPointF.Reflect

Declaration: `function Reflect(const normal: TPointF) : TPointF`

Visibility: `public`

77.7.18 TPointF.MidPoint

Declaration: `function MidPoint(const b: TPointF) : TPointF`

Visibility: `public`

77.7.19 TPointF.PointInCircle

Declaration: `class function PointInCircle(const pt: TPointF; const center: TPointF;
radius: single) : Boolean; Static
class function PointInCircle(const pt: TPointF; const center: TPointF;
radius: Integer) : Boolean; Static`

Visibility: `public`

77.7.20 TPointF.Zero

Declaration: `class function Zero : TPointF; Static`

Visibility: public

77.7.21 TPointF.Angle

Declaration: `function Angle(const b: TPointF) : Single`

Visibility: public

77.7.22 TPointF.AngleCosine

Declaration: `function AngleCosine(const b: TPointF) : single`

Visibility: public

77.7.23 TPointF.CrossProduct

Declaration: `function CrossProduct(const apt: TPointF) : Single`

Visibility: public

77.7.24 TPointF.Normalize

Declaration: `function Normalize : TPointF`

Visibility: public

77.7.25 TPointF.Create

Declaration: `class function Create(const ax: Single; const ay: Single) : TPointF
; Overload; Static
class function Create(const apt: TPoint) : TPointF; Overload; Static`

Visibility: public

77.7.26 TPointF.equal(TPointF,TPointF):Boolean

Declaration: `TPointF.class operator =(const apt1: TPointF; const apt2: TPointF)
: Boolean`

Visibility: public

77.7.27 TPointF.notequal(TPointF,TPointF):Boolean

Declaration: `TPointF.class operator <>(const apt1: TPointF; const apt2: TPointF)
: Boolean`

Visibility: public

77.7.28 TPointF.add(TPointF,TPointF):TPointF

Declaration: TPointF.class operator +(const apt1: TPointF; const apt2: TPointF)
: TPointF

Visibility: public

77.7.29 TPointF.subtract(TPointF,TPointF):TPointF

Declaration: TPointF.class operator -(const apt1: TPointF; const apt2: TPointF)
: TPointF

Visibility: public

77.7.30 TPointF.negative(TPointF):TPointF

Declaration: TPointF.class operator -(const apt1: TPointF) : TPointF

Visibility: public

77.7.31 TPointF.multiply(TPointF,TPointF):TPointF

Declaration: TPointF.class operator *(const apt1: TPointF; const apt2: TPointF)
: TPointF

Visibility: public

77.7.32 TPointF.multiply(TPointF,single):TPointF

Declaration: TPointF.class operator *(const apt1: TPointF; afactor: single) : TPointF

Visibility: public

77.7.33 TPointF.multiply(single,TPointF):TPointF

Declaration: TPointF.class operator *(afactor: single; const apt1: TPointF) : TPointF

Visibility: public

77.7.34 TPointF.divide(TPointF,single):TPointF

Declaration: TPointF.class operator /(const apt1: TPointF; afactor: single) : TPointF

Visibility: public

77.7.35 TPointF.assign(TPoint):TPointF

Declaration: TPointF.class operator :=(const apt: TPoint) : TPointF

Visibility: public

77.7.36 TPointF.power(TPointF,TPointF):Single

Declaration: TPointF.class operator ******(const apt1: TPointF; const apt2: TPointF)
: Single

Visibility: public

77.8 TRect

```

TRect = packed record
private
    function getHeight : LongInt;
    function
        getLocation : TPoint;
    function getSize : TSize;
    function getWidth
        : LongInt;
    procedure setHeight (AValue: LongInt);
    procedure setSize
        (AValue: TSize);
    procedure setWidth(AValue: LongInt);
public
    Create
    ;
    TRect.class operator =(L: TRect; R: TRect) : Boolean;
    TRect
        .class operator <>(L: TRect; R: TRect) : Boolean;
    TRect.class operator
        +(L: TRect; R: TRect) : TRect;
    TRect.class operator *(L: TRect
        ; R: TRect) : TRect;
    class function Empty : TRect; Static;
    procedure
        NormalizeRect;
    function IsEmpty : Boolean;
    Contains;
    function
        IntersectsWith(R: TRect) : Boolean;
    Intersect;
    Union;
    Offset
    ;
    SetLocation;
    Inflate;
    function CenterPoint : TPoint;
    SplitRect
    ;
    property Height : LongInt;
    property Width : LongInt;
    property
        Size : TSize;
    property Location : TPoint;
case LongInt of

```

```

0: (
  public
  Left : LongInt;
  Top : LongInt;
  Right : LongInt;
  Bottom
    : LongInt;
);
1: (
public
  TopLeft : TPoint;
  BottomRight : TPoint
  ;
);
2: (
public
  Vector : TArray4IntegerType;
);
end

```

TRect defines a rectangle in a discrete plane. It is described by the horizontal (`left`, `right`) or vertical (`top`, `Bottom`) positions (in pixels) of the edges, or, alternatively, by the coordinates of the top left (`TopLeft`) and bottom right (`BottomRight`) corners.

77.8.1 Method overview

Page	Method	Description
2038	<code>add(TRect,TRect):TRect</code>	
2039	<code>CenterPoint</code>	
2039	<code>Contains</code>	
2038	<code>Create</code>	
2038	<code>Empty</code>	
2038	<code>equal(TRect,TRect):Boolean</code>	
2039	<code>Inflate</code>	
2039	<code>Intersect</code>	
2039	<code>IntersectsWith</code>	
2038	<code>IsEmpty</code>	
2038	<code>multiply(TRect,TRect):TRect</code>	
2038	<code>NormalizeRect</code>	
2038	<code>notequal(TRect,TRect):Boolean</code>	
2039	<code>Offset</code>	
2039	<code>SetLocation</code>	
2040	<code>SplitRect</code>	
2039	<code>Union</code>	

77.8.2 Property overview

Page	Properties	Access	Description
2040	<code>Height</code>	rw	
2040	<code>Location</code>	rw	
2040	<code>Size</code>	rw	
2040	<code>Width</code>	rw	

77.8.3 TRect.Create

Declaration: constructor Create(Origin: TPoint)
 constructor Create(Origin: TPoint; AWidth: LongInt; AHeight: LongInt)
 constructor Create(ALeft: LongInt; ATop: LongInt; ARight: LongInt;
 ABottom: LongInt)
 constructor Create(P1: TPoint; P2: TPoint; Normalize: Boolean)
 constructor Create(R: TRect; Normalize: Boolean)

Visibility: public

77.8.4 TRect.equal(TRect,TRect):Boolean

Declaration: TRect.class operator =(L: TRect; R: TRect) : Boolean

Visibility: public

77.8.5 TRect.notequal(TRect,TRect):Boolean

Declaration: TRect.class operator <>(L: TRect; R: TRect) : Boolean

Visibility: public

77.8.6 TRect.add(TRect,TRect):TRect

Declaration: TRect.class operator +(L: TRect; R: TRect) : TRect

Visibility: public

77.8.7 TRect.multiply(TRect,TRect):TRect

Declaration: TRect.class operator *(L: TRect; R: TRect) : TRect

Visibility: public

77.8.8 TRect.Empty

Declaration: class function Empty : TRect; Static

Visibility: public

77.8.9 TRect.NormalizeRect

Declaration: procedure NormalizeRect

Visibility: public

77.8.10 TRect.IsEmpty

Declaration: function IsEmpty : Boolean

Visibility: public

77.8.11 TRect.Contains

Declaration: function Contains(Pt: TPoint) : Boolean
 function Contains(R: TRect) : Boolean

Visibility: public

77.8.12 TRect.IntersectsWith

Declaration: function IntersectsWith(R: TRect) : Boolean

Visibility: public

77.8.13 TRect.Intersect

Declaration: class function Intersect(R1: TRect; R2: TRect) : TRect; Static
 procedure Intersect(R: TRect)

Visibility: public

77.8.14 TRect.Union

Declaration: class function Union(R1: TRect; R2: TRect) : TRect; Static
 procedure Union(R: TRect)
 class function Union(const Points: Array of TPoint) : TRect; Static

Visibility: public

77.8.15 TRect.Offset

Declaration: procedure Offset(DX: LongInt; DY: LongInt)
 procedure Offset(DP: TPoint)

Visibility: public

77.8.16 TRect.SetLocation

Declaration: procedure SetLocation(X: LongInt; Y: LongInt)
 procedure SetLocation(P: TPoint)

Visibility: public

77.8.17 TRect.Inflate

Declaration: procedure Inflate(DX: LongInt; DY: LongInt)
 procedure Inflate(DL: LongInt; DT: LongInt; DR: LongInt; DB: LongInt)

Visibility: public

77.8.18 TRect.CenterPoint

Declaration: function CenterPoint : TPoint

Visibility: public

77.8.19 TRect.SplitRect

Declaration: `function SplitRect (SplitType: TSplitRectType; ASize: LongInt) : TRect`
`function SplitRect (SplitType: TSplitRectType; Percent: Double) : TRect`

Visibility: public

77.8.20 TRect.Height

Declaration: `Property Height : LongInt`

Visibility: public

Access: Read,Write

77.8.21 TRect.Width

Declaration: `Property Width : LongInt`

Visibility: public

Access: Read,Write

77.8.22 TRect.Size

Declaration: `Property Size : TSize`

Visibility: public

Access: Read,Write

77.8.23 TRect.Location

Declaration: `Property Location : TPoint`

Visibility: public

Access: Read,Write

77.9 TRectF

```
TRectF = packed record
private
    function GetLocation : TPointF;
    function GetSize : TSizeF;
    procedure SetSize(AValue: TSizeF);
    function GetHeight : Single;
    function GetWidth : Single;
    procedure
        SetHeight(AValue: Single);
    procedure SetWidth(AValue: Single);
public
    Create;
    TRectF.class operator =(L: TRectF; R: TRectF)
```

```

: Boolean;
TRectF.class operator <>(L: TRectF; R: TRectF) : Boolean
;
TRectF.class operator +(L: TRectF; R: TRectF) : TRectF;
TRectF
.class operator *(L: TRectF; R: TRectF) : TRectF;
TRectF.class operator
:=(const arc: TRect) : TRectF;
class function Empty : TRectF;
Static;
Intersect;
Union;
function Ceiling : TRectF;
function
CenterAt(const Dest: TRectF) : TRectF;
function CenterPoint : TPointF
;
Contains;
function EqualsTo(const R: TRectF; const Epsilon:
Single) : Boolean;
function Fit(const Dest: TRectF) : Single;
FitInto;
function IntersectsWith(R: TRectF) : Boolean;
function
IsEmpty : Boolean;
function PlaceInto(const Dest: TRectF;
const AHorzAlign: THorzRectAlign;
const AVertAlign: TVertRectAlign) : TRectF;
function Round
: TRect;
function SnapToPixel(AScale: Single; APlaceBetweenPixels
: Boolean)
: TRectF;

function Truncate :
TRect;
Inflate;
procedure Normalize;
procedure NormalizeRect
;
Offset;
SetLocation;
property Width : Single;
property Height
: Single;
property Size : TSizeF;
property Location : TPointF
;
case Integer of
0: (
public
Left : Single;
Top : Single;
Right
: Single;

```

```

    Bottom : Single;
);
1: (
public
    TopLeft : TPointF;
    BottomRight : TPointF;
);
end

```

77.9.1 Method overview

Page	Method	Description
2043	add(TRectF,TRectF):TRectF	
2043	assign(TRect):TRectF	
2044	Ceiling	
2044	CenterAt	
2044	CenterPoint	
2044	Contains	
2042	Create	
2043	Empty	
2043	equal(TRectF,TRectF):Boolean	
2044	EqualsTo	
2044	Fit	
2044	FitInto	
2045	Inflate	
2043	Intersect	
2045	IntersectsWith	
2045	IsEmpty	
2043	multiply(TRectF,TRectF):TRectF	
2045	Normalize	
2046	NormalizeRect	
2043	notequal(TRectF,TRectF):Boolean	
2046	Offset	
2045	PlaceInto	
2045	Round	
2046	SetLocation	
2045	SnapToPixel	
2045	Truncate	
2044	Union	

77.9.2 Property overview

Page	Properties	Access	Description
2046	Height	rw	
2046	Location	rw	
2046	Size	rw	
2046	Width	rw	

77.9.3 TRectF.Create

Declaration: constructor Create(Origin: TPointF)
 constructor Create(Origin: TPointF; AWidth: Single; AHeight: Single)

```

constructor Create(ALeft: Single; ATop: Single; ARight: Single;
                  ABottom: Single)
constructor Create(P1: TPointF; P2: TPointF; Normalize: Boolean)
constructor Create(R: TRectF; Normalize: Boolean)
constructor Create(R: TRect; Normalize: Boolean)

```

Visibility: public

77.9.4 TRectF.equal(TRectF,TRectF):Boolean

Declaration: TRectF.class operator =(L: TRectF; R: TRectF) : Boolean

Visibility: public

77.9.5 TRectF.notequal(TRectF,TRectF):Boolean

Declaration: TRectF.class operator <>(L: TRectF; R: TRectF) : Boolean

Visibility: public

77.9.6 TRectF.add(TRectF,TRectF):TRectF

Declaration: TRectF.class operator +(L: TRectF; R: TRectF) : TRectF

Visibility: public

77.9.7 TRectF.multiply(TRectF,TRectF):TRectF

Declaration: TRectF.class operator *(L: TRectF; R: TRectF) : TRectF

Visibility: public

77.9.8 TRectF.assign(TRect):TRectF

Declaration: TRectF.class operator :=(const arc: TRect) : TRectF

Visibility: public

77.9.9 TRectF.Empty

Declaration: class function Empty : TRectF; Static

Visibility: public

77.9.10 TRectF.Intersect

Declaration: class function Intersect(R1: TRectF; R2: TRectF) : TRectF; Static
 procedure Intersect(R: TRectF)

Visibility: public

77.9.11 TRectF.Union

Declaration: `class function Union(const Points: Array of TPointF) : TRectF; Static`
`class function Union(R1: TRectF; R2: TRectF) : TRectF; Static`
`procedure Union(const r: TRectF)`

Visibility: public

77.9.12 TRectF.Ceiling

Declaration: `function Ceiling : TRectF`

Visibility: public

77.9.13 TRectF.CenterAt

Declaration: `function CenterAt(const Dest: TRectF) : TRectF`

Visibility: public

77.9.14 TRectF.CenterPoint

Declaration: `function CenterPoint : TPointF`

Visibility: public

77.9.15 TRectF.Contains

Declaration: `function Contains(Pt: TPointF) : Boolean`
`function Contains(R: TRectF) : Boolean`

Visibility: public

77.9.16 TRectF.EqualsTo

Declaration: `function EqualsTo(const R: TRectF; const Epsilon: Single) : Boolean`

Visibility: public

77.9.17 TRectF.Fit

Declaration: `function Fit(const Dest: TRectF) : Single`

Visibility: public

77.9.18 TRectF.FitInto

Declaration: `function FitInto(const Dest: TRectF) : TRectF; Overload`
`function FitInto(const Dest: TRectF; out Ratio: Single) : TRectF`
`; Overload`

Visibility: public

77.9.27 TRectF.NormalizeRect

Declaration: `procedure NormalizeRect`

Visibility: `public`

77.9.28 TRectF.Offset

Declaration: `procedure Offset(const dx: Single; const dy: Single)`
`procedure Offset(DP: TPointF)`

Visibility: `public`

77.9.29 TRectF.SetLocation

Declaration: `procedure SetLocation(P: TPointF)`
`procedure SetLocation(X: Single; Y: Single)`

Visibility: `public`

77.9.30 TRectF.Width

Declaration: `Property Width : Single`

Visibility: `public`

Access: `Read,Write`

77.9.31 TRectF.Height

Declaration: `Property Height : Single`

Visibility: `public`

Access: `Read,Write`

77.9.32 TRectF.Size

Declaration: `Property Size : TSizeF`

Visibility: `public`

Access: `Read,Write`

77.9.33 TRectF.Location

Declaration: `Property Location : TPointF`

Visibility: `public`

Access: `Read,Write`

77.10 TSize

```

TSize = packed record
public
  cx : LongInt;
  cy : LongInt;
  function
    Add(const asz: TSize) : TSize;
  function Distance(const asz: TSize
    ) : Double;
  function IsZero : Boolean;
  function Subtract(const
    asz: TSize) : TSize;
  TSize.class operator =(const asz1: TSize;
    const asz2: TSize) : Boolean;
  TSize.class operator <>(const asz1
    : TSize; const asz2: TSize) : Boolean;
  TSize.class operator +(const
    asz1: TSize; const asz2: TSize) : TSize;
  TSize.class operator
    -(const asz1: TSize; const asz2: TSize) : TSize;
  property Width
    : LongInt;
  property Height : LongInt;
end

```

TSize is a type to describe the size of a rectangular area, where *cx* is the width, *cy* is the height (in pixels) of the rectangle.

77.10.1 Method overview

Page	Method	Description
2047	Add	
2048	add(TSize,TSize):TSize	
2048	Distance	
2048	equal(TSize,TSize):Boolean	
2048	IsZero	
2048	notequal(TSize,TSize):Boolean	
2048	Subtract	
2048	subtract(TSize,TSize):TSize	

77.10.2 Property overview

Page	Properties	Access	Description
2049	Height	rw	
2048	Width	rw	

77.10.3 TSize.Add

Declaration: `function Add(const asz: TSize) : TSize`

Visibility: `public`

77.10.4 TSize.Distance

Declaration: `function Distance(const asz: TSize) : Double`

Visibility: public

77.10.5 TSize.IsZero

Declaration: `function IsZero : Boolean`

Visibility: public

77.10.6 TSize.Subtract

Declaration: `function Subtract(const asz: TSize) : TSize`

Visibility: public

77.10.7 TSize.equal(TSize,TSize):Boolean

Declaration: `TSize.class operator =(const asz1: TSize; const asz2: TSize) : Boolean`

Visibility: public

77.10.8 TSize.notequal(TSize,TSize):Boolean

Declaration: `TSize.class operator <>(const asz1: TSize; const asz2: TSize) : Boolean`

Visibility: public

77.10.9 TSize.add(TSize,TSize):TSize

Declaration: `TSize.class operator +(const asz1: TSize; const asz2: TSize) : TSize`

Visibility: public

77.10.10 TSize.subtract(TSize,TSize):TSize

Declaration: `TSize.class operator -(const asz1: TSize; const asz2: TSize) : TSize`

Visibility: public

77.10.11 TSize.Width

Declaration: `Property Width : LongInt`

Visibility: public

Access: Read,Write

77.10.12 TSize.Height

Declaration: Property Height : LongInt

Visibility: public

Access: Read,Write

77.11 TSizeF

```

TSizeF = packed record
public
  cx : Single;
  cy : Single;
  Add
  ;
  function Distance(const asz: TSizeF) : Single;
  function IsZero
  : Boolean;
  Subtract;
  function SwapDimensions : TSizeF;
  function
  Scale(afactor: Single) : TSizeF;
  function Ceiling : TSize;
  function
  Truncate : TSize;
  function Floor : TSize;
  function Round : TSize
  ;
  function &Length : Single;
  Create;
  TSizeF.class operator
  =(const asz1: TSizeF; const asz2: TSizeF)
  : Boolean;
  TSizeF.class operator <>(const asz1: TSizeF; const
  asz2: TSizeF)
  : Boolean;

  TSizeF.class
  operator +(const asz1: TSizeF; const asz2: TSizeF) : TSizeF;
  TSizeF
  .class operator -(const asz1: TSizeF; const asz2: TSizeF) : TSizeF
  ;
  TSizeF.class operator -(const asz1: TSizeF) : TSizeF;
  TSizeF
  .class operator *(const asz1: TSizeF; afactor: single) : TSizeF;
  TSizeF.class operator *(afactor: single; const asz1: TSizeF) : TSizeF
  ;
  TSizeF.class operator :=(const apt: TPointF) : TSizeF;
  TSizeF
  .class operator :=(const asz: TSize) : TSizeF;
  TSizeF.class operator
  :=(const asz: TSizeF) : TPointF;
  property Width : Single;
  property

```

```

    Height : Single;
end

```

77.11.1 Method overview

Page	Method	Description
2050	Add	
2052	add(TSizeF,TSizeF):TSizeF	
2053	assign(TPointF):TSizeF	
2053	assign(TSize):TSizeF	
2053	assign(TSizeF):TPointF	
2051	Ceiling	
2052	Create	
2050	Distance	
2052	equal(TSizeF,TSizeF):Boolean	
2051	Floor	
2050	IsZero	
2051	Length	
2052	multiply(single,TSizeF):TSizeF	
2052	multiply(TSizeF,single):TSizeF	
2052	negative(TSizeF):TSizeF	
2052	notequal(TSizeF,TSizeF):Boolean	
2051	Round	
2051	Scale	
2051	Subtract	
2052	subtract(TSizeF,TSizeF):TSizeF	
2051	SwapDimensions	
2051	Truncate	

77.11.2 Property overview

Page	Properties	Access	Description
2053	Height	rw	
2053	Width	rw	

77.11.3 TSizeF.Add

Declaration: `function Add(const asz: TSize) : TSizeF`
`function Add(const asz: TSizeF) : TSizeF`

Visibility: public

77.11.4 TSizeF.Distance

Declaration: `function Distance(const asz: TSizeF) : Single`

Visibility: public

77.11.5 TSizeF.IsZero

Declaration: `function IsZero : Boolean`

Visibility: public

77.11.6 TSizeF.Subtract

Declaration: `function Subtract(const asz: TSizeF) : TSizeF`
`function Subtract(const asz: TSize) : TSizeF`

Visibility: public

77.11.7 TSizeF.SwapDimensions

Declaration: `function SwapDimensions : TSizeF`

Visibility: public

77.11.8 TSizeF.Scale

Declaration: `function Scale(afactor: Single) : TSizeF`

Visibility: public

77.11.9 TSizeF.Ceiling

Declaration: `function Ceiling : TSize`

Visibility: public

77.11.10 TSizeF.Truncate

Declaration: `function Truncate : TSize`

Visibility: public

77.11.11 TSizeF.Floor

Declaration: `function Floor : TSize`

Visibility: public

77.11.12 TSizeF.Round

Declaration: `function Round : TSize`

Visibility: public

77.11.13 TSizeF.Length

Declaration: `function &Length : Single`

Visibility: public

77.11.14 TSizeF.Create

Declaration: `class function Create(const ax: Single; const ay: Single) : TSizeF
; Overload; Static
class function Create(const asz: TSize) : TSizeF; Overload; Static`

Visibility: public

77.11.15 TSizeF.equal(TSizeF,TSizeF):Boolean

Declaration: `TSizeF.class operator =(const asz1: TSizeF; const asz2: TSizeF)
: Boolean`

Visibility: public

77.11.16 TSizeF.notequal(TSizeF,TSizeF):Boolean

Declaration: `TSizeF.class operator <>(const asz1: TSizeF; const asz2: TSizeF)
: Boolean`

Visibility: public

77.11.17 TSizeF.add(TSizeF,TSizeF):TSizeF

Declaration: `TSizeF.class operator +(const asz1: TSizeF; const asz2: TSizeF) : TSizeF`

Visibility: public

77.11.18 TSizeF.subtract(TSizeF,TSizeF):TSizeF

Declaration: `TSizeF.class operator -(const asz1: TSizeF; const asz2: TSizeF) : TSizeF`

Visibility: public

77.11.19 TSizeF.negative(TSizeF):TSizeF

Declaration: `TSizeF.class operator -(const asz1: TSizeF) : TSizeF`

Visibility: public

77.11.20 TSizeF.multiply(TSizeF,single):TSizeF

Declaration: `TSizeF.class operator *(const asz1: TSizeF; afactor: single) : TSizeF`

Visibility: public

77.11.21 TSizeF.multiply(single,TSizeF):TSizeF

Declaration: `TSizeF.class operator *(afactor: single; const asz1: TSizeF) : TSizeF`

Visibility: public

77.11.22 TSizeF.assign(TPointF):TSizeF

Declaration: `TSizeF.class operator :=(const apt: TPointF) : TSizeF`

Visibility: public

77.11.23 TSizeF.assign(TSize):TSizeF

Declaration: `TSizeF.class operator :=(const asz: TSize) : TSizeF`

Visibility: public

77.11.24 TSizeF.assign(TSizeF):TPointF

Declaration: `TSizeF.class operator :=(const asz: TSizeF) : TPointF`

Visibility: public

77.11.25 TSizeF.Width

Declaration: `Property Width : Single`

Visibility: public

Access: Read,Write

77.11.26 TSizeF.Height

Declaration: `Property Height : Single`

Visibility: public

Access: Read,Write

77.12 IClassFactory**77.12.1 Description**

`IClassFactory` is defined for Delphi/Kylix compatibility and should not be used.

77.12.2 Method overview

Page	Method	Description
2053	<code>CreateInstance</code>	Create a new instance of an interface.
2054	<code>LockServer</code>	Lock ActiveX server object.

77.12.3 IClassFactory.CreateInstance

Synopsis: Create a new instance of an interface.

Declaration: `function CreateInstance(const unkOuter: IUnknown; const riid: TGuid;
out vObject) : HRESULT`

Visibility: default

Description: `IClassFactory.CreateInstance` is defined for Delphi/Kylix compatibility and should not be used.

77.12.4 IClassFactory.LockServer

Synopsis: Lock ActiveX server object.

Declaration: `function LockServer(fLock: LongBool) : HRESULT`

Visibility: default

Description: `IClassFactory.LockServer` is defined for Delphi/Kylix compatibility and should not be used.

77.13 ISequentialStream

77.13.1 Description

`ISequentialStream` is the interface for streams which only support sequential reading of chunks of data. It is defined for Delphi/Kylix compatibility and should not be used.

See also: `IStream` ([2055](#))

77.13.2 Method overview

Page	Method	Description
2054	Read	Read data from the stream.
2054	Write	Write data to the stream.

77.13.3 ISequentialStream.Read

Synopsis: Read data from the stream.

Declaration: `function Read(pv: Pointer; cb: DWORD; pcbRead: PDWord) : HRESULT`

Visibility: default

Description: `Read` reads `cbCount` bytes from the stream into the memory pointed to by `pv` and returns the number of bytes read in `pcbread`. The result is zero for success or an error code.

See also: `ISequentialStream.Write` ([2054](#))

77.13.4 ISequentialStream.Write

Synopsis: Write data to the stream.

Declaration: `function Write(pv: Pointer; cb: DWORD; pcbWritten: PDWord) : HRESULT`

Visibility: default

Description: `Write` writes `cbCount` bytes from the memory pointed to by `pv` to the stream and returns the number of bytes written in `pcbwritten`. The result is zero for success or an error code.

See also: `ISequentialStream.Read` ([2054](#))

77.14 IStream

77.14.1 Description

An abstract interface for an external (non pascal) stream, as defined in Microsoft COM interfaces

77.14.2 Method overview

Page	Method	Description
2057	Clone	Clone the stream instance.
2056	Commit	Commit data to the stream.
2055	CopyTo	Copy data from one stream to another.
2056	LockRegion	Lock a region of bytes in the stream.
2056	Revert	Revert changes.
2055	Seek	Set the stream position.
2055	SetSize	Set the stream size.
2057	Stat	return information about the stream.
2056	UnlockRegion	Unlocks a previously locked region of bytes in the stream.

77.14.3 IStream.Seek

Synopsis: Set the stream position.

Declaration: `function Seek(dlibMove: Largeint; dwOrigin: DWORD;
out libNewPosition: LargeUint) : HRESULT`

Visibility: default

Description: `Seek` sets the stream position at `dlibMove` bytes from `dwOrigin` (one of the `SEEK_*` constants) and returns the new absolute position in `libNewPosition`. The function returns zero on success, or an error code.

Errors: On error, a nonzero exit code is returned.

77.14.4 IStream.SetSize

Synopsis: Set the stream size.

Declaration: `function SetSize(libNewSize: LargeUint) : HRESULT`

Visibility: default

Description: `SetSize` sets the size of the stream to `libNewSize` bytes, if the stream allows it. On success, zero is returned.

Errors: On error, a nonzero exit code is returned.

77.14.5 IStream.CopyTo

Synopsis: Copy data from one stream to another.

Declaration: `function CopyTo(stm: IStream; cb: LargeUint; out cbRead: LargeUint;
out cbWritten: LargeUint) : HRESULT`

Visibility: default

Description: `CopyTo` copies `cb` bytes from the stream to target stream `stm`. `cbRead` returns how many bytes were read from the stream, `cbwrite` returns how many bytes were actually written to the destination stream. The function returns zero on success.

Errors: On error, a nonzero exit code is returned.

77.14.6 IStream.Commit

Synopsis: Commit data to the stream.

Declaration: `function Commit(grfCommitFlags: DWORD) : HRESULT`

Visibility: default

Description: `Commit` commits the data in the stream to the underlying medium. `Flags` is a set of options to control the commit operation (see MSDN for the possible flags).

Errors: On error, a nonzero exit code is returned.

77.14.7 IStream.Revert

Synopsis: Revert changes.

Declaration: `function Revert : HRESULT`

Visibility: default

Description: `Revert` reverts all changes that were done to a transacted stream, i.e. all changes since the last commit. The function returns zero on success.

Errors: On error, a nonzero exit code is returned.

77.14.8 IStream.LockRegion

Synopsis: Lock a region of bytes in the stream.

Declaration: `function LockRegion(libOffset: LargeUint; cb: LargeUint;
dwLockType: DWORD) : HRESULT`

Visibility: default

Description: `LockRegion` locks a region of the storage, starting at `libOffset`, for `cbCount` bytes. The applied lock is of type `dwLockType`. The function returns zero if the lock was successfully applied.

Errors: On error, a nonzero exit code is returned.

77.14.9 IStream.UnlockRegion

Synopsis: Unlocks a previously locked region of bytes in the stream.

Declaration: `function UnlockRegion(libOffset: LargeUint; cb: LargeUint;
dwLockType: DWORD) : HRESULT`

Visibility: default

Description: `UnlockRegion` removes the lock on a region of the storage, starting at `libOffset`, for `cbCount` bytes. The lock must be of type `dwLockType`. The function returns zero if the lock was successfully removed.

Errors: On error, a nonzero exit code is returned.

77.14.10 IStream.Stat

Synopsis: return information about the stream.

Declaration: `function Stat(out statstg: TStatStg; grfStatFlag: DWORD) : HRESULT`

Visibility: default

Description: `Stat` returns information about the stream in `statstg`, taking into account the flags in `grfStatFlag` (one of the `STATFLAG_` constants). The function returns zero if the call was successful.

Errors: On error, a nonzero exit code is returned.

77.14.11 IStream.Clone

Synopsis: Clone the stream instance.

Declaration: `function Clone(out stm: IStream) : HRESULT`

Visibility: default

Description: `Clone` returns an independent but initially equal copy of the stream in `stm`. The function returns zero if the call was successful.

Errors: On error, a nonzero exit code is returned.

77.15 TBitConverter

77.15.1 Method overview

Page	Method	Description
2057	<code>From</code>	
2058	<code>InTo</code>	
2057	<code>UnsafeFrom</code>	
2058	<code>UnsafeInTo</code>	

77.15.2 TBitConverter.UnsafeFrom

Declaration: `class procedure UnsafeFrom<T>(const ASrcValue: T;
var ADestination: Array of Byte;
AOffset: Integer); Static`

Visibility: default

77.15.3 TBitConverter.From

Declaration: `class procedure From<T>(const ASrcValue: T;
var ADestination: Array of Byte;
AOffset: Integer); Static`

Visibility: default

77.15.4 TBitConverter.UnsafeInTo

Declaration: `class function UnsafeInTo<T>(const ASource: Array of Byte;
AOffset: Integer) : T; Static`

Visibility: default

77.15.5 TBitConverter.InTo

Declaration: `class function InTo<T>(const ASource: Array of Byte; AOffset: Integer)
: T; Static`

Visibility: default

Chapter 78

Reference for unit 'TypeInfo'

78.1 Used units

Table 78.1: Used units by unit 'TypeInfo'

Name	Page
System	1335
sysutils	1609

78.2 Overview

The `TypeInfo` unit contains many routines which can be used for the querying of the Run-Time Type Information (RTTI) which is generated by the compiler for classes that are compiled under the `{ $M+ }` switch. This information can be used to retrieve or set property values for published properties for totally unknown classes. In particular, it can be used to stream classes. The `TPersistent` class in the `Classes` unit is compiled in the `{ $M+ }` state and serves as the base class for all classes that need to be streamed.

The unit should be compatible to the Delphi unit with the same name.

The examples in this chapter use a `rttiobj` auxiliary unit, which contains an object that has a published property for all supported types. It also contains some auxiliary routines and definitions. This unit is included in the documentation sources, in the directory `typinfex`.

```
Unit rttiobj;

{$Mode Delphi}
{$M+}

Interface

uses TypeInfo;

Const TypeName : Array [TTypeKind] of string[15] =
    ('Unknown', 'Integer', 'Char', 'Enumeration',
     'Float', 'Set', 'Method', 'ShortString', 'LongString',
```

```

        'AnsiString', 'WideString', 'Variant', 'Array', 'Record',
        'Interface', 'Class', 'Object', 'WideChar', 'Bool', 'Int64', 'QWord',
        'DynamicArray', 'RawInterface', 'ProcVar', 'UString', 'UChar',
        'Helper', 'File', 'ClassRef', 'Pointer');

Const OrdinalTypes = [tkInteger, tkChar, tkEnumeration, tkbool];

Type
    TMyEnum = (meFirst, meSecond, meThird);
    TMyEnums = Set of TMyEnum;
    TNotifyEvent = Procedure (Sender : TObject) of object;

    TMyTestObject = Class (TObject)
    Private
        FBoolean : Boolean;
        FByte : Byte;
        FChar : Char;
        FWord : Word;
        FInteger : Integer;
        FLongint : Longint;
        FCardinal : Cardinal;
        FReal : Real;
        FExtended : Extended;
        FMyEnum : TMyEnum;
        FAnsiString : AnsiString;
        FObj : TObject;
        FNotifyEvent : TNotifyEvent;
        FSetField : TMyEnums;
//      FInt64Field : Int64;
        FInt64Field : Integer;
        FStored : Boolean;
        Function GetBoolean : Boolean;
        Function GetByte : Byte;
        Function GetChar : Char;
        Function GetWord : Word;
        Function GetInteger : Integer;
        Function GetLongint : Longint;
        Function GetCardinal : Cardinal;
        Function GetReal : Real;
        Function GetExtended : Extended;
        Function GetAnsiString : AnsiString;
        Function GetMyEnum : TMyEnum;
        Procedure SetBoolean ( Value : Boolean);
        Procedure SetByte ( Value : Byte );
        Procedure SetChar ( Value : Char );
        Procedure SetWord ( Value : Word );
        Procedure SetInteger ( Value : Integer );
        Procedure SetLongint ( Value : Longint );
        Procedure SetCardinal ( Value : Cardinal );
        Procedure SetReal ( Value : Real );
        Procedure SetExtended ( Value : Extended );
        Procedure SetAnsiString ( Value : AnsiString );
        Procedure SetMyEnum ( Value : TMyEnum );

```

```

Function GetVirtualBoolean : Boolean; virtual;
Function GetVirtualByte : Byte; virtual;
Function GetVirtualChar : Char; virtual;
Function GetVirtualWord : Word; virtual;
Function GetVirtualInteger : Integer; virtual;
Function GetVirtualLongint : Longint; virtual;
Function GetVirtualCardinal : Cardinal; virtual;
Function GetVirtualReal : Real; virtual;
Function GetVirtualExtended : Extended; virtual;
Function GetVirtualAnsiString : AnsiString; virtual;
Function GetVirtualMyEnum : TMyEnum; virtual;
Procedure SetVirtualBoolean ( Value : Boolean); virtual;
Procedure SetVirtualByte ( Value : Byte ); virtual;
Procedure SetVirtualChar ( Value : Char ); virtual;
Procedure SetVirtualWord ( Value : Word ); virtual;
Procedure SetVirtualInteger ( Value : Integer ); virtual;
Procedure SetVirtualLongint ( Value : Longint ); virtual;
Procedure SetVirtualCardinal ( Value : Cardinal ); virtual;
Procedure SetVirtualReal ( Value : Real ); virtual;
Procedure SetVirtualExtended ( Value : Extended ); virtual;
Procedure SetVirtualAnsiString ( Value : AnsiString ); virtual;
Procedure SetVirtualMyEnum ( Value : TMyEnum ); virtual;
Function GetStaticStored : Boolean;
Function GetVirtualStored : Boolean;virtual;
Public
  Constructor Create;
  Destructor Destroy;override;
  Procedure Notify;
Published
  Property ObjField: TObject read FObj write FObj;
  Property SetField : TMyEnums Read FSetField Write FSetField;
  Property NotifyEvent : TNotifyEvent Read FNotifyEvent Write FNotifyEvent;
//   Property Int64Field : Int64 Read FInt64Field Write FInt64Field;
  Property Int64Field : Integer Read FInt64Field Write FInt64Field;
  Property BooleanField : Boolean Read FBoolean Write FBoolean;
  Property ByteField : Byte Read FByte Write FByte;
  Property CharField : Char Read FChar Write FChar;
  Property WordField : Word Read FWord Write FWord;
  Property IntegerField : Integer Read FInteger Write FInteger;
  Property LongintField : Longint Read FLongint Write FLongint;
  Property CardinalField : Cardinal Read FCardinal Write FCardinal;
  Property RealField : Real Read FReal Write FReal;
  Property ExtendedField : Extended Read FExtended Write FExtended;
  Property AnsiStringField : AnsiString Read FAnsiString Write FAnsiString;
  Property MyEnumField : TMyEnum Read FMyEnum Write FMyEnum;
  Property BooleanMethod : Boolean Read GetBoolean Write SetBoolean;
  Property ByteMethod : Byte Read GetByte Write SetByte;
  Property CharMethod : Char Read GetChar Write SetChar;
  Property WordMethod : Word Read GetWord Write SetWord;
  Property IntegerMethod : Integer Read GetInteger Write SetInteger;
  Property LongintMethod : Longint Read GetLongint Write SetLongint;
  Property CardinalMethod : Cardinal Read GetCardinal Write SetCardinal;
  Property RealMethod : Real Read GetReal Write SetReal;
  Property ExtendedMethod : Extended Read GetExtended Write SetExtended;

```

```

Property AnsiStringMethod : AnsiString Read GetAnsiString Write SetAnsiString;
Property MyEnumMethod : TMyEnum Read GetMyEnum Write SetMyEnum;
Property BooleanVirtualMethod : Boolean Read GetVirtualBoolean Write SetVirtualBoolean;
Property ByteVirtualMethod : Byte Read GetVirtualByte Write SetVirtualByte;
Property CharVirtualMethod : Char Read GetVirtualChar Write SetVirtualChar;
Property WordVirtualMethod : Word Read GetVirtualWord Write SetVirtualWord;
Property IntegerVirtualMethod : Integer Read GetVirtualInteger Write SetVirtualInteger;
Property LongintVirtualMethod : Longint Read GetVirtualLongint Write SetVirtualLongint;
Property CardinalVirtualMethod : Cardinal Read GetVirtualCardinal Write SetVirtualCardinal;
Property RealVirtualMethod : Real Read GetVirtualReal Write SetVirtualReal;
Property ExtendedVirtualMethod : Extended Read GetVirtualExtended Write SetVirtualExtended;
Property AnsiStringVirtualMethod : AnsiString Read GetVirtualAnsiString Write SetVirtualAnsiString;
Property MyEnumVirtualMethod : TMyEnum Read GetVirtualMyEnum Write SetVirtualMyEnum;
Property StoredIntegerConstFalse : Longint Read FLongint Stored False;
Property StoredIntegerConstTrue : Longint Read FLongint Stored True;
Property StoredIntegerField : Longint Read FLongint Stored FStored;
Property StoredIntegerMethod : Longint Read FLongint Stored GetStaticStored;
Property StoredIntegerVirtualMethod : Longint Read FLongint Stored GetVirtualStored;
end;

TNamedObject = Class(TObject)
Private
    FObjectName : AnsiString;
Public
    Property ObjectName : AnsiString Read FObjectName Write FObjectName;
end;

Const
    MyEnumNames : Array[TMyEnum] of string = ('meFirst','meSecond','meThird');

Procedure PrintObject ( Obj: TMyTestObject);

Implementation

Constructor TMyTestObject.Create;

begin
    FBoolean:=true;
    FByte:=1;           {      : Byte; }
    FChar:='B';         {      : Char; }
    FWord:=3;           { : Word;      }
    FInteger:=4;         { : Integer;   }
    FLongint:=5;         { : Longint;   }
    FCardinal:=6;        { : Cardinal;  }
    FReal:=7.0;          {      : Real;  }
    FExtended :=8.0;     { Extended; }
    FMyEnum:=methird;    { TMyEnum; }
    FAnsiString:='this is an AnsiString';
end;

Destructor TMyTestObject.Destroy;

begin
    Inherited Destroy;

```

```
end;

Function TMyTestObject.GetBoolean : boolean;

begin
    Result:=FBoolean;
end;

Function TMyTestObject.GetByte : Byte;

begin
    Result:=FByte;
end;

Function TMyTestObject.GetChar : Char;
begin
    Result:=FChar;
end;

Function TMyTestObject.GetWord : Word;
begin
    Result:=FWord;
end;

Function TMyTestObject.GetInteger : Integer;
begin
    Result:=FInteger;
end;

Function TMyTestObject.GetLongint : Longint;
begin
    Result:=FLongint;
end;

Function TMyTestObject.GetCardinal : Cardinal;
begin
    Result:=FCardinal;
end;

Function TMyTestObject.GetReal : Real;
begin
    Result:=FReal;
end;

Function TMyTestObject.GetExtended : Extended;
begin
    Result:=FExtended;
end;

Function TMyTestObject.GetAnsiString : AnsiString;
begin
    Result:=FAnsiString;
end;
```



```
Function TMyTestObject.GetMyEnum : TMyEnum;
begin
    Result:=FMyEnum;
end;

Procedure TMyTestObject.Setboolean ( Value : boolean );
begin
    Fboolean:=Value;
end;

Procedure TMyTestObject.SetByte ( Value : Byte );
begin
    FByte:=Value;
end;

Procedure TMyTestObject.SetChar ( Value : Char );
begin
    FChar:=Value;
end;

Procedure TMyTestObject.SetWord ( Value : Word );
begin
    FWord:=Value;
end;

Procedure TMyTestObject.SetInteger ( Value : Integer );
begin
    FInteger:=Value;
end;

Procedure TMyTestObject.SetLongint ( Value : Longint );
begin
    FLongint:=Value;
end;

Procedure TMyTestObject.SetCardinal ( Value : Cardinal );
begin
    FCardinal:=Value;
end;

Procedure TMyTestObject.SetReal ( Value : Real );
begin
    FReal:=Value;
end;

Procedure TMyTestObject.SetExtended ( Value : Extended );
begin
    FExtended:=Value;
end;

Procedure TMyTestObject.SetAnsiString ( Value : AnsiString );
begin
    FAnsiString:=Value;
```

```
end;

Procedure TMyTestObject.SetMyEnum ( Value : TMyEnum );
begin
    FMyEnum:=Value;
end;

Function TMyTestObject.GetVirtualBoolean : boolean;

begin
    Result:=FBoolean;
end;

Function TMyTestObject.GetVirtualByte : Byte;

begin
    Result:=FByte;
end;

Function TMyTestObject.GetVirtualChar : Char;
begin
    Result:=FChar;
end;

Function TMyTestObject.GetVirtualWord : Word;
begin
    Result:=FWord;
end;

Function TMyTestObject.GetVirtualInteger : Integer;
begin
    Result:=FInteger;
end;

Function TMyTestObject.GetVirtualLongint : Longint;
begin
    Result:=FLongint;
end;

Function TMyTestObject.GetVirtualCardinal : Cardinal;
begin
    Result:=FCardinal;
end;

Function TMyTestObject.GetVirtualReal : Real;
begin
    Result:=FReal;
end;

Function TMyTestObject.GetVirtualExtended : Extended;
begin
    Result:=FExtended;
end;
```

```
Function TMyTestObject.GetVirtualAnsiString : AnsiString;
begin
    Result:=FAnsiString;
end;

Function TMyTestObject.GetVirtualMyEnum : TMyEnum;
begin
    Result:=FMyEnum;
end;

Procedure TMyTestObject.SetVirtualboolean ( Value : boolean );
begin
    Fboolean:=Value;
end;

Procedure TMyTestObject.SetVirtualByte ( Value : Byte );
begin
    FByte:=Value;
end;

Procedure TMyTestObject.SetVirtualChar ( Value : Char );
begin
    FChar:=Value;
end;

Procedure TMyTestObject.SetVirtualWord ( Value : Word );
begin
    FWord:=Value;
end;

Procedure TMyTestObject.SetVirtualInteger ( Value : Integer );
begin
    FInteger:=Value;
end;

Procedure TMyTestObject.SetVirtualLongint ( Value : Longint );
begin
    FLongint:=Value;
end;

Procedure TMyTestObject.SetVirtualCardinal ( Value : Cardinal );
begin
    FCardinal:=Value;
end;

Procedure TMyTestObject.SetVirtualReal ( Value : Real );
begin
    FReal:=Value;
end;

Procedure TMyTestObject.SetVirtualExtended ( Value : Extended );
begin
    FExtended:=Value;
```

```

end;

Procedure TMyTestObject.SetVirtualAnsiString ( Value : AnsiString );
begin
    FAnsiString:=Value;
end;

Procedure TMyTestObject.SetVirtualMyEnum ( Value : TMyEnum );
begin
    FMyEnum:=Value;
end;

Function TMyTestObject.GetStaticStored : Boolean;

begin
    Result:=False;
end;

Function TMyTestObject.GetVirtualStored : Boolean;

begin
    Result:=False;
end;

Procedure TMyTestObject.Notify;

begin
    If Assigned(FNotifyEvent) then
        FNotifyEvent(Self)
    else
        Writeln('Error : No notifyevent assigned');
end;

Procedure PrintObject ( Obj: TMyTestObject);

begin
    With Obj do
        begin
            Writeln ('Field properties :');
            Writeln ('Property booleanField      : ',booleanField);
            Writeln ('Property ByteField        : ',ByteField);
            Writeln ('Property CharField         : ',CharField);
            Writeln ('Property WordField         : ',WordField);
            Writeln ('Property IntegerField      : ',IntegerField);
            Writeln ('Property LongintField      : ',LongintField);
            Writeln ('Property CardinalField     : ',CardinalField);
            Writeln ('Property RealField         : ',RealField);
            Writeln ('Property ExtendedField     : ',ExtendedField);
            Writeln ('Property AnsiStringField   : ',AnsiStringField);
            Writeln ('Property MyEnumField       : ',ord(MyEnumField));
            Writeln ('Method properties :');
            Writeln ('Property booleanMethod     : ',BooleanMethod);
            Writeln ('Property ByteMethod        : ',ByteMethod);

```

```

Writeln ('Property CharMethod      : ',CharMethod);
Writeln ('Property WordMethod      : ',WordMethod);
Writeln ('Property IntegerMethod   : ',IntegerMethod);
Writeln ('Property LongintMethod   : ',LongintMethod);
Writeln ('Property CardinalMethod   : ',CardinalMethod);
Writeln ('Property RealMethod       : ',RealMethod);
Writeln ('Property ExtendedMethod   : ',ExtendedMethod);
Writeln ('Property AnsiStringMethod : ',AnsiStringMethod);
Writeln ('Property MyEnumMethod      : ',ord(MyEnumMethod));
Writeln ('VirtualMethod properties :');
Writeln ('Property booleanVirtualMethod : ',BooleanVirtualMethod);
Writeln ('Property ByteVirtualMethod      : ',ByteVirtualMethod);
Writeln ('Property CharVirtualMethod      : ',CharVirtualMethod);
Writeln ('Property WordVirtualMethod      : ',WordVirtualMethod);
Writeln ('Property IntegerVirtualMethod   : ',IntegerVirtualMethod);
Writeln ('Property LongintVirtualMethod   : ',LongintVirtualMethod);
Writeln ('Property CardinalVirtualMethod : ',CardinalVirtualMethod);
Writeln ('Property RealVirtualMethod      : ',RealVirtualMethod);
Writeln ('Property ExtendedVirtualMethod : ',ExtendedVirtualMethod);
Writeln ('Property AnsiStringVirtualMethod : ',AnsiStringVirtualMethod);
Writeln ('Property MyEnumVirtualMethod    : ',ord(MyEnumVirtualMethod));
end;
end;

Procedure DumpMem ( PL : PByte );

Var I,j : longint;

begin
  For I:=1 to 16 do
    begin
      Write ((I-1)*16:3,' :');
      For J:=1 to 10 do
        begin
          If (PL^>31) and (PL^<129) then
            Write(' ',Char(PL^))
          else
            Write (PL^:3);
          Write (' ');
          inc(pl);
        end;
      writeln;
    end;
end;

Function ProcType (PP : Byte) : String;

begin
  Case PP and 3 of
    ptfield   : Result:='from Field';
    ptstatic  : Result:='with static method';
    ptVirtual : Result:='with virtual method';
    ptconst   : Result:='with Const';
  end;
end;

```

```

    end;
end;

end.

```

78.3 Auxiliary functions.

Other typinfo related functions.

Table 78.2:

Name	Description
GetEnumName (2084)	Get an enumerated type element name
GetEnumValue (2086)	Get ordinal number of an enumerated type, based on the name.
GetEnumNameCount (2085)	Get number of elements in an enumerated type.
GetTypeData (2099)	Skip type name and return a pointer to the type data
SetToString (2109)	Convert a set to its string representation
StringToSet (2111)	Convert a string representation of a set to a set

78.4 Getting or setting property values.

Functions to set or set a property's value.

Table 78.3:

Name	Description
GetEnumProp (2085)	Return the value of an enumerated type property
GetFloatProp (2086)	Return the value of a float property
GetInt64Prop (2087)	Return the value of an Int64 property
GetMethodProp (2088)	Return the value of a procedural type property
GetObjectProp (2090)	Return the value of an object property
GetOrdProp (2092)	Return the value of an ordinal type property
GetProperty (2095)	Return the value of a property as a variant
GetSetProp (2097)	Return the value of a set property
GetStrProp (2098)	Return the value of a string property
GetWideStrProp (2100)	Return the value of a widestring property
GetVariantProp (2099)	Return the value of a variant property
SetEnumProp (2105)	Set the value of an enumerated type property
SetFloatProp (2105)	Set the value of a float property
SetInt64Prop (2105)	Set the value of an Int64 property
SetMethodProp (2106)	Set the value of a procedural type property
SetObjectProp (2107)	Set the value of an object property
SetOrdProp (2107)	Set the value of an ordinal type property
SetPropValue (2108)	Set the value of a property through a variant
SetSetProp (2109)	Set the value of a set property
SetStrProp (2109)	Set the value of a string property
SetWideStrProp (2111)	Set the value of a widestring property
SetVariantProp (2111)	Set the value of a variant property

78.5 Examining published property information.

Functions for retrieving or examining property information

Table 78.4:

Name	Description
FindPropInfo (2082)	Getting property type information, With error checking.
GetPropInfo (2093)	Getting property type information, No error checking.
GetPropInfos (2093)	Find property information of a certain kind
GetObjectPropClass (2091)	Return the declared class of an object property
GetPropList (2094)	Get a list of all published properties
IsPublishedProp (2100)	Is a property published
IsStoredProp (2101)	Is a property stored
PropIsType (2102)	Is a property of a certain kind
PropType (2103)	Return the type of a property

78.6 Constants, types and variables

78.6.1 Constants

```
BooleanIdents : Array[Boolean] of string = ('False', 'True')
```

Names for boolean values.

```
DotSep : string = '.'
```

Name separator character.

```
OnGetPropValue : TGetPropValue = Nil
```

This callback is set by the variants unit to enable reading of properties as a variant. If set, it is called by the GetPropValue (2095) function.

```
OnGetVariantprop : TGetVariantProp = Nil
```

This callback is set by the variants unit to enable reading of variant properties. If set, it is called by the GetVariantProp (2099) function.

```
OnSetPropValue : TSetPropValue = Nil
```

This callback is set by the variants unit to enable writing of properties as a variant. If set, it is called by the SetPropValue (2108) function.

```
OnSetVariantprop : TSetVariantProp = Nil
```

This callback is set by the variants unit to enable writing of variant properties. If set, it is called by the GetVariantProp (2099) function.

```
ptConst = 3
```

Constant used in access method.

```
ptField = 0
```

Property access directly from field.

```
ptStatic = 1
```

Property access via static method.

```
ptVirtual = 2
```

Property access via virtual method.

```
tkAny = [Low(TTypeKind) .. High(TTypeKind)]
```

Any property type.

```
tkArray = System.tkArray
```

Array property.

```
tkAString = System.tkAString
```

Ansistring property.

```
tkBool = System.tkBool
```

Boolean property.

```
tkChar = System.tkChar
```

Char property.

```
tkClass = System.tkClass
```

Class property.

```
tkClassRef = System.tkClassRef
```

```
tkDynArray = System.tkDynArray
```

Dynamical array property.

```
tkEnumeration = System.tkEnumeration
```

Enumeration type property.

```
tkFile = System.tkFile
```


`tkFloat = System.tkFloat`

Float property.

`tkHelper = System.tkHelper`

`tkInt64 = System.tkInt64`

Int64 property.

`tkInteger = System.tkInteger`

Integer property.

`tkInterface = System.tkInterface`

Interface property.

`tkInterfaceRaw = System.tkInterfaceRaw`

Raw interface property.

`tkLString = System.tkLString`

Longstring property.

`tkMethod = System.tkMethod`

Method property.

`tkMethods = [tkMethod]`

Only method properties. (event handlers).

`tkObject = System.tkObject`

Object property.

`tkPointer = System.tkPointer`

`tkProcedure = tkProcVar`

Procedure kind.

`tkProcVar = System.tkProcVar`

`tkProperties = tkAny - tkMethods - [tkUnknown]`

Real properties. (not methods).

`tkQWord = System.tkQWord`

QWord property.

`tkRecord = System.tkRecord`

Record property.

`tkSet = System.tkSet`

Set property.

`tkSString = System.tkSString`

Shortstring property.

`tkString = tkSString`

Alias for the `tsSString` enumeration value.

`tkUChar = System.tkUChar`

`tkUnknown = System.tkUnknown`

Unknown property type.

`tkUString = System.tkUString`

`tkVariant = System.tkVariant`

Variant property.

`tkWChar = System.tkWChar`

Widechar property.

`tkWString = System.tkWString`

Widestring property.

78.6.2 Types

`PClassData = ^TClassData`

`PClassData` is a pointer to a `TClassData` ([2113](#)) record.

`PInitManagedField = ^TInitManagedField`

`PInitManagedField` is a pointer to a `TInitManagedField` ([2076](#)) record.

`PInterfaceData = ^TInterfaceData`

`PInterfaceData` is a pointer to a `TInterfaceData` (2114) record.

`PInterfaceRawData = ^TInterfaceRawData`

Pointer to a `TInterfaceRawData` record.

`PIntfMethodEntry = ^TIntfMethodEntry`

`PIntfMethodEntry` is a pointer to a `TIntfMethodEntry` (2117) record.

`PIntfMethodTable = ^TIntfMethodTable`

`PIntfMethodTable` is a pointer to a `TIntfMethodTable` (2119) record.

`PManagedField = ^TManagedField`

`PManagedField` is a pointer to `TManagedField` (2120). It is used to describe automatically managed fields in records when the type kind is `tkRecord`.

`PParameterLocation = ^TParameterLocation`

`PParameterLocation` defines a pointer to a `TParameterLocation` (2121) record.

`PParameterLocations = ^TParameterLocations`

`PParameterLocations` points to a `TParameterLocations` (2122) record.

`PProcedureParam = ^TProcedureParam`

`PProcedureParam` is a pointer to `TProcedureParam`. It is used in `TProcedureSignature` (2124).

`PProcedureSignature = ^TProcedureSignature`

`PPropData = ^TPropData`

`PPropData` is a pointer to a `TPropData` (2125) record.

`PPropInfo = ^TPropInfo`

Pointer to `TPropInfo` (2127) record.

`PPropList = ^TPropList`

Pointer to `TPropList` (2078).

`PTypeInfo = ^PTypeInfo`

Pointer to PTypeInfo (2075) pointer.

PRecInitData = ^TRecInitData

PRecInitData is a pointer to a TRecInitData (2079) record.

PRecOpOffsetTable = ^TRecOpOffsetTable

PRecOpOffsetTable is a Pointer to a TRecOpOffsetTable (2079) record.

PTypeData = ^TTypeData

Pointer to TTypeData (2143) record.

PTypeInfo = ^TTypeInfo

Pointer to TTypeInfo (2080) record.

PVmtFieldClassTab = ^TVmtFieldClassTab

PVmtFieldClassTab points to a TVmtFieldClassTab (2081) record.

PVmtFieldEntry = ^TVmtFieldEntry

Pointer to #rtl.typinfo.TVmtFieldEntry (2146) type.

PVmtFieldTable = ^TVmtFieldTable

Pointer to #rtl.typinfo.TVmtFieldTable (2147) type.

PVmtMethodEntry = ^TVmtMethodEntry

PVmtMethodEntry is a Pointer to TVmtMethodEntry (2081) record.

PVmtMethodParam = ^TVmtMethodParam

PVmtMethodParam is a pointer to a TVmtMethodParam (2148) record.

PVmtMethodTable = ^TVmtMethodTable

PVmtMethodTable is a pointer to a TVmtMethodTable (2149) record.

ShortStringBase = string

ShortStringBase is the base definition of a short string.

TCallConv = (ccReg, ccCdecl, ccPascal, ccStdCall, ccSafeCall, ccCppdecl
,
 ccFar16, ccOldFPCCall, ccInternProc, ccSysCall, ccSoftFloat
,
 ccMWPascal)

Table 78.5: Enumeration values for type TCallConv

Value	Explanation
ccCdecl	Cdecl calling convention.
ccCppdecl	Cppdecl calling convention.
ccFar16	Far16 calling convention (Delphi compatibility).
ccInternProc	InternProc calling convention (compiler internal).
ccMWPPascal	MWPPascal (MetroWerks Pascal) calling convention.
ccOldFPCCall	OldFPCCall calling convention (deprecated).
ccPascal	Pascal calling convention.
ccReg	Register calling convention.
ccSafeCall	SafeCall calling convention.
ccSoftFloat	Softfloat calling convention.
ccStdCall	stdcall calling convention.
ccSysCall	SysCall calling convention.

TCallConv is a type describing the calling convention used by a method. It contains an element for all supported calling conventions.

```
TFloatType = (ftSingle, ftDouble, ftExtended, ftComp, ftCurr)
```

Table 78.6: Enumeration values for type TFloatType

Value	Explanation
ftComp	Comp-type float.
ftCurr	Currency-type float.
ftDouble	Double-sized float.
ftExtended	Extended-size float.
ftSingle	Single-sized float.

The size of a float type.

```
TGetPropValue = function(Instance: TObject; PropInfo: PPropInfo;
    PreferStrings: Boolean) : Variant
```

The callback function must return the property with name `PropName` of instance `Instance`. If `PreferStrings` is true, it should favour converting the property to a string value. The function needs to return the variant with the property value.

```
TGetVariantProp = function(Instance: TObject; PropInfo: PPropInfo
    )
    : Variant
```

The callback function must return the variant property with name `PropName` of instance `Instance`.

```
TInitManagedField = TManagedField
```

TInitManagedField is an alias for TManagedField ([2120](#)).

```
TIntfFlag = (ifHasGuid,ifDispInterface,ifDispatch,ifHasStrGUID)
```

Table 78.7: Enumeration values for type TIntfFlag

Value	Explanation
ifDispatch	Interface is a dispatch interface.
ifDispInterface	Interface is a dual dispatch interface.
ifHasGuid	Interface has GUID identifier.
ifHasStrGUID	Interface has a string GUID identifier.

Type of interface.

```
TIntfFlags = Set of TIntfFlag
```

Set of TIntfFlag (2077).

```
TIntfFlagsBase = Set of TIntfFlag
```

Set of TIntfFlag (2077).

```
TMethodKind = (mkProcedure,mkFunction,mkConstructor,mkDestructor,
  mkClassProcedure,mkClassFunction,mkClassConstructor
  ,
  mkClassDestructor,mkOperatorOverload)
```

Table 78.8: Enumeration values for type TMethodKind

Value	Explanation
mkClassConstructor	Class constructor method.
mkClassDestructor	Class destructor method.
mkClassFunction	Class function.
mkClassProcedure	Class procedure.
mkConstructor	Class constructor.
mkDestructor	Class Destructor.
mkFunction	Function method.
mkOperatorOverload	Operator overloader.
mkProcedure	Procedure method.

Method type description.

```
TOrdType = (otSByte,otUByte,otSWord,otUWord,otSLong,otULong,otSQWord
  ,
  otUQWord)
```

Table 78.9: Enumeration values for type TOrdType

Value	Explanation
otSByte	Signed byte.
otSLong	Signed longint.
otSQWord	Signed QWord.
otSWord	Signed word.
otUByte	Unsigned byte.
otULong	Unsigned longint (Cardinal).
otUQWord	Unsigned QWord.
otUWord	Unsigned word.

If the property is an ordinal type, then TOrdType determines the size and sign of the ordinal type:

```
TParamFlag = (pfVar, pfConst, pfArray, pfAddress, pfReference, pfOut,
  pfConstRef, pfHidden, pfHigh, pfSelf, pfVmt, pfResult)
```

Table 78.10: Enumeration values for type TParamFlag

Value	Explanation
pfAddress	Parameter is passed by address.
pfArray	Parameter is an array parameter.
pfConst	Parameter is a const parameter (i.e. cannot be modified).
pfConstRef	Constref parameter type.
pfHidden	hidden parameter type.
pfHigh	High bound parameter type.
pfOut	Parameter is a string parameter.
pfReference	Parameter is passed by reference.
pfResult	Result location parameter.
pfSelf	Self parameter type.
pfVar	Parameter is a var parameter (passed by reference).
pfVmt	VMT parameter type.

TParamFlag describes a parameter.

```
TParamFlags = Set of TParamFlag
```

The kind of parameter for a method.

```
TProcInfoProc = procedure(PropInfo: PPropInfo) of object
```

Property info callback method.

```
TPropList = Array[0..65535] of PPropInfo
```

Array of property information pointers.

```
TRecInitData = packed record
```

```

public
  Terminator : Pointer;
  Size
  : Integer;
  InitOffsetOp : PRecOpOffsetTable;
  ManagementOp : Pointer
  ;
  ManagedFieldCount : Integer;
end

```

TRecInitData describes the generated RTTI info with which you can initialize a record that contains managed fields.

```

TRecOpOffsetEntry = packed record
public
  ManagementOperator : CodePointer
  ;
  FieldOffset : SizeUInt;
end

```

TRecOpOffsetEntry describes management operator info for a record.

```

TRecOpOffsetTable = packed record
public
  Count : LongWord;
  Entries
  : Array[0..0] of TRecOpOffsetEntry;
end

```

TRecOpOffsetTable is a table of TRecOpOffsetEntry (2079) records with management operator info for the managed fields of a record.

```
TRegisterType = (Invalid, Int, FP, MMX, MultiMedia, Special, Address)
```

Table 78.11: Enumeration values for type TRegisterType

Value	Explanation
Address	Address register.
FP	Floating-point register.
Int	Integer register.
Invalid	Invalid register.
MMX	MMX register.
MultiMedia	Multimedia register.
Special	Special register.

TRegisterType describes the type of the CPU register.

```

TSetPropValue = procedure (Instance: TObject; PropInfo: PPropInfo;
  const Value: Variant)

```

The callback function must set the property with name PropName of instance Instance to Value.


```
TSetVariantProp = procedure (Instance: TObject; PropInfo: PPropInfo
;
                                const Value: Variant)
```

The callback function must set the variant property with name `PropName` of instance to `Value`.

```
TSubRegister = (None, Lo, Hi, Word, DWord, QWord, FloatSingle, FloatDouble
,
                FloatQuad, MultiMediaSingle, MultiMediaDouble,
                MultiMediaWhole, MultiMediaX, MultiMediaY)
```

Table 78.12: Enumeration values for type `TSubRegister`

Value	Explanation
DWord	DWord size.
FloatDouble	Double-sized float value.
FloatQuad	Quadruple-sized float value.
FloatSingle	Single-sized float value.
Hi	High.
Lo	Low.
MultiMediaDouble	Multimedia register double-sized float.
MultiMediaSingle	Multimedia register single-sized float.
MultiMediaWhole	Multimedia register whole-sized.
MultiMediaX	Multimedia register X position.
MultiMediaY	Multimedia register Y position.
None	None.
QWord	QWord size.
Word	Word size.

`TSubRegister` describes CPU sub-register sizes or locations.

```
TTypeInfo = record
public
    Kind : TTypeKind;
    Name : ShortString
;
end
```

The `TypeInfo` function returns a pointer to a `TTypeInfo` record.

Note that the `Name` field is stored with as much bytes as needed to store the name, it is not padded to 255 characters. The type data immediately follows the `TTypeInfo` record as a `TTypeData` (2143) record.

```
TTypeKind = System.TTypeKind
```

`TTypeKind` was moved to the system unit, see `#rtl.system.TTypeKind` (1401)

```
TTypeKinds = Set of TTypeKind
```

Set of `TTypeKind` (2080) enumeration.

```

TVmtFieldClassTab = packed record
public
  Count : Word;
  ClassRef
    : Array[0..0] of PClass;
end

```

TVmtFieldClassTab describes the fields of class type. A pointer to this table is present in TVmtFieldTable (2147)

```

TVmtMethodEntry = packed record
public
  Name : PShortString;
  CodeAddress
    : CodePointer;
end

```

TVmtMethodEntry is a description of the VMT info generated for a method of a class. This is different from interface method RTTI info, which is described by TIntfMethodEntry (2117)

78.7 Procedures and functions

78.7.1 AddEnumElementAliases

Synopsis: Define aliases for enumerated elements.

Declaration: `procedure AddEnumElementAliases(aTypeInfo: PTypeInfo; const aNames: Array of string; aStartValue: Integer)`

Visibility: default

Description: AddEnumElementAliases defines aliases for enumerated values of type aTypeInfo. The aliases are specified in aNames, and are applied as of the element with ordinal value aStartIndex. The aliases will be used in GetEnumValue (2086)

See also: RemoveEnumElementAliases (2104), GetEnumValue (2086), GetEnumeratedAliasValue (2084)

78.7.2 AlignPTypeInfo

Synopsis: Align TTypeInfo info.

Declaration: `function AlignPTypeInfo(p: Pointer) : Pointer`

Visibility: default

Description: AlignPTypeInfo will align the pointer p on a correct boundary for TParamFlags (2078) data for the current platform and returns the resulting address.

See also: align (2059), AlignTypeData (2082), AlignPTypeInfo (2081)

78.7.3 AlignTParamFlags

Synopsis: Align TParamFlags info.

Declaration: `function AlignTParamFlags(p: Pointer) : Pointer`

Visibility: default

Description: `AlignTParamFlags` will align the pointer `p` on a correct boundary for TParamFlags (2078) data for the current platform and returns the resulting address.

See also: `align` (2059), `AlignTypeData` (2082), `AlignPTypeInfo` (2081)

78.7.4 AlignTypeData

Synopsis: Align type data.

Declaration: `function AlignTypeData(p: Pointer) : Pointer`

Visibility: default

Description: `AlignTypeData` will align the pointer `p` on a correct boundary for type data for the current platform and returns the resulting address.

See also: `align` (2059), `AlignTParamFlags` (2082), `AlignPTypeInfo` (2081)

78.7.5 DerefTypeInfoPtr

Synopsis: Dereference type info pointer.

Declaration: `function DerefTypeInfoPtr(Info: PTypeInfo) : PTypeInfo`

Visibility: default

Description: `DerefTypeInfoPtr` can be used to dereference the `Info` type info pointer if need be: in FPC version 3.0, this pointer does not need dereferencing, but in FPC version 3.1, the pointer must be dereferenced. This is useful for code that must work in version 3.0 and later.

Errors: The function handles nil values well.

See also: `PTypeInfo` (2075)

78.7.6 FindPropInfo

Synopsis: Return property information by property name.

```
Declaration: function FindPropInfo(Instance: TObject; const PropName: string)
                : PPropInfo
function FindPropInfo(Instance: TObject; const PropName: string;
                AKinds: TTypeKinds) : PPropInfo
function FindPropInfo(AClass: TClass; const PropName: string)
                : PPropInfo
function FindPropInfo(AClass: TClass; const PropName: string;
                AKinds: TTypeKinds) : PPropInfo
```

Visibility: default

Description: `FindPropInfo` examines the published property information of a class and returns a pointer to the property information for property `PropName`. The class to be examined can be specified in one of two ways:

AClass a class pointer.

Instance an instance of the class to be investigated.

If the property does not exist, a `EPropertyError` exception will be raised. The `GetPropInfo` (2093) function has the same function as the `FindPropInfo` function, but returns `Nil` if the property does not exist.

Errors: Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: `GetPropInfo` (2093), `GetPropList` (2094), `GetPropInfos` (2093)

Listing: `./examples/typinfex/ex14.pp`

Program `example13;`

{ This program demonstrates the FindPropInfo function }

{ \$mode objfpc }

uses

`rttiobj, typinfo, sysutils;`

Var

`O : TMyTestObject;`

`PT : PTypeData;`

`PI : PPropInfo;`

`I, J : Longint;`

`PP : PPropList;`

`prl : PPropInfo;`

begin

`O := TMyTestObject.Create;`

`PI := FindPropInfo(O, 'BooleanField');`

`Writeln('FindPropInfo(Instance, BooleanField) : ', PI^.Name);`

`PI := FindPropInfo(O.ClassType, 'ByteField');`

`Writeln('FindPropInfo(Class, ByteField) : ', PI^.Name);`

`Write ('FindPropInfo(Class, NonExistingProp) : ');`

Try

`PI := FindPropInfo(O, 'NonExistingProp');`

except

On `E: Exception` **do**

`Writeln('Caught exception "', E.ClassName, '" with message : ', E.Message);`

end;

`O.Free;`

end.

78.7.7 GetDynArrayProp

Synopsis: Get the value of a dynamic array valued property.

Declaration: `function GetDynArrayProp(Instance: TObject; const PropName: string)
: Pointer`

```
function GetDynArrayProp (Instance: TObject; PropInfo: PPropInfo)
    : Pointer
```

Visibility: default

Description: `GetDynArrayProp` returns the value of a dynamic array valued property of the object `Instance` as a pointer. The property to read can be specified using the name `PropName` or the property info `PropInfo`.

See also: `SetDynArrayProp` ([2104](#))

78.7.8 GetEnumeratedAliasValue

Synopsis: Return ordinal value for an enumerated value.

```
Declaration: function GetEnumeratedAliasValue (aTypeInfo: PTypeInfo;
    const aName: string) : Integer
```

Visibility: default

Description: `GetEnumeratedAliasValue` will return the ordinal value of the alias `aName` for type `aTypeInfo`.

See also: `AddEnumElementAliases` ([2081](#)), `RemoveEnumElementAliases` ([2104](#)), `GetEnumValue` ([2086](#))

78.7.9 GetEnumName

Synopsis: Return name of enumeration constant.

```
Declaration: function GetEnumName (TypeInfo: PTypeInfo; Value: Integer) : string
```

Visibility: default

Description: `GetEnumName` scans the type information for the enumeration type described by `TypeInfo` and returns the name of the enumeration constant for the element with ordinal value equal to `Value`.

If `Value` is out of range, the first element of the enumeration type is returned. The result is returned in the case that was used in the declaration. (In earlier versions of FPC, the name was lowercased).

This can be used in combination with `GetOrdProp` to stream a property of an enumerated type.

Errors: No check is done to determine whether `TypeInfo` really points to the type information for an enumerated type.

See also: `GetOrdProp` ([2092](#)), `GetEnumValue` ([2086](#))

Listing: `./examples/typinfex/ex9.pp`

```
program example9;
```

```
{ This program demonstrates the GetEnumName, GetEnumValue functions }
```

```
{ $mode objfpc }
```

```
uses rttiobj, typinfo;
```

```
Var
```

```
  O : TMyTestObject;
```

```
  TI : PTypeInfo;
```

```
begin
```

```
O:=TMyTestObject.Create;
TI:=GetPropInfo(O,'MyEnumField')^.PropType;
WriteLn('GetEnumName : ',GetEnumName(TI,Ord(O.MyEnumField)));
WriteLn('GetEnumValue(mefirst) : ',GetEnumName(TI,GetEnumValue(TI,'mefirst')));
O.Free;
end.
```

78.7.10 GetEnumNameCount

Synopsis: Return number of names in an enumerated type.

Declaration: `function GetEnumNameCount(enum1: PTypeInfo) : SizeInt`

Visibility: default

Description: `GetEnumNameCount` returns the number of values (names) in the enumerated type, described by `enum1`

Errors: No checking is done to see whether `Enum1` is really type information of an enumerated type.

See also: `GetEnumValue` (2086), `GetEnumName` (2084)

78.7.11 GetEnumProp

Synopsis: Return the value of an enumeration type property.

Declaration: `function GetEnumProp(Instance: TObject; const PropName: string) : string`
`function GetEnumProp(Instance: TObject; const PropInfo: PPropInfo)`
`: string`

Visibility: default

Description: `GetEnumProp` returns the value of an property of an enumerated type and returns the name of the enumerated value for the object `Instance`. The property whose value must be returned can be specified by its property info in `PropInfo` or by its name in `PropName`

Errors: No check is done to determine whether `PropInfo` really points to the property information for an enumerated type. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: `SetEnumProp` (2105), `GetOrdProp` (2092), `GetStrProp` (2098), `GetInt64Prop` (2087), `GetMethodProp` (2088), `GetSetProp` (2097), `GetObjectProp` (2090), `GetEnumProp` (2085)

Listing: `./examples/typinfex/ex2.pp`

```
program example2;

{ This program demonstrates the GetEnumProp function }

{$mode objfpc}

uses rtlibobj,typinfo;

Var
  O : TMyTestObject;
  PI : PPropInfo;
  TI : PTypeInfo;
```

```

begin
  O:=TMyTestObject.Create;
  PI:=GetPropInfo(O,'MyEnumField');
  TI:=PI^.PropType;
  Writeln('Enum property : ');
  Writeln('Value : ',GetEnumName(TI,Ord(O.MyEnumField)));
  Writeln('Get (name) : ',GetEnumProp(O,'MyEnumField'));
  Writeln('Get (propinfo) : ',GetEnumProp(O,PI));
  SetEnumProp(O,'MyEnumField','meFirst');
  Writeln('Set (name,meFirst) : ',GetEnumName(TI,Ord(O.MyEnumField)));
  SetEnumProp(O,PI,'meSecond');
  Writeln('Set (propinfo,meSecond) : ',GetEnumName(TI,Ord(O.MyEnumField)));
  O.Free;
end.

```

78.7.12 GetEnumValue

Synopsis: Get ordinal value for enumerated type by name.

Declaration: `function GetEnumValue (TypeInfo: PTypeInfo; const Name: string) : Integer`

Visibility: default

Description: `GetEnumValue` scans the type information for the enumeration type described by `TypeInfo` and returns the ordinal value for the element in the enumerated type that has identifier `Name`. The identifier is searched in a case-insensitive manner.

This can be used to set the value of enumerated properties from a stream.

For an example, see `GetEnumName` (2084).

Errors: If `Name` is not found in the list of enumerated values, then -1 is returned. No check is done whether `TypeInfo` points to the type information for an enumerated type.

See also: `GetEnumName` (2084), `SetOrdProp` (2107)

78.7.13 GetFloatProp

Synopsis: Return value of floating point property.

Declaration: `function GetFloatProp (Instance: TObject; PropInfo: PPropInfo) : Extended`
`function GetFloatProp (Instance: TObject; const PropName: string)`
`: Extended`

Visibility: default

Description: `GetFloatProp` returns the value of the float property described by `PropInfo` or with name `Propname` for the object `Instance`. All float types are converted to extended.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid float property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: `SetFloatProp` (2105), `GetOrdProp` (2092), `GetStrProp` (2098), `GetInt64Prop` (2087), `GetMethodProp` (2088), `GetSetProp` (2097), `GetObjectProp` (2090), `GetEnumProp` (2085)

Listing: `./examples/typinfex/ex4.pp`

```

program example4;

{ This program demonstrates the GetFloatProp function }

{$mode objfpc}

uses rttiobj,typinfo;

Var
  O : TMyTestObject;
  PI : PPropInfo;

begin
  O:=TMyTestObject.Create;
  Writeln('Real property : ');
  PI:=GetPropInfo(O,'RealField');
  Writeln('Value : ',O.RealField);
  Writeln('Get (name) : ',GetFloatProp(O,'RealField'));
  Writeln('Get (propinfo) : ',GetFloatProp(O,PI));
  SetFloatProp(O,'RealField',system.Pi);
  Writeln('Set (name,pi) : ',O.RealField);
  SetFloatProp(O,PI,exp(1));
  Writeln('Set (propinfo,e) : ',O.RealField);
  Writeln('Extended property : ');
  PI:=GetPropInfo(O,'ExtendedField');
  Writeln('Value : ',O.ExtendedField);
  Writeln('Get (name) : ',GetFloatProp(O,'ExtendedField'));
  Writeln('Get (propinfo) : ',GetFloatProp(O,PI));
  SetFloatProp(O,'ExtendedField',system.Pi);
  Writeln('Set (name,pi) : ',O.ExtendedField);
  SetFloatProp(O,PI,exp(1));
  Writeln('Set (propinfo,e) : ',O.ExtendedField);
  O.Free;
end.

```

78.7.14 GetInt64Prop

Synopsis: return value of an Int64 property.

Declaration: `function GetInt64Prop(Instance: TObject; PropInfo: PPropInfo) : Int64`
`function GetInt64Prop(Instance: TObject; const PropName: string) : Int64`

Visibility: default

Exception: Publishing of Int64 properties is not yet supported by Free Pascal. This function is provided for Delphi compatibility only at the moment.

`GetInt64Prop` returns the value of the property of type `Int64` that is described by `PropInfo` or with name `Propname` for the object `Instance`.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid `Int64` property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception

See also: [SetInt64Prop \(2105\)](#), [GetOrdProp \(2092\)](#), [GetStrProp \(2098\)](#), [GetFloatProp \(2086\)](#), [GetMethodProp \(2088\)](#), [GetSetProp \(2097\)](#), [GetObjectProp \(2090\)](#), [GetEnumProp \(2085\)](#)

Listing: ./examples/typinfex/ex15.pp

```

program example15;

{ This program demonstrates the GetInt64Prop function }

{$mode objfpc}

uses rttiobj, typinfo;

Var
  O : TMyTestObject;
  PI : PPropInfo;

begin
  O:=TMyTestObject.Create;
  Writeln('Int64 property : ');
  PI:=GetPropInfo(O,'Int64Field');
  Writeln('Value : ',O.Int64Field);
  Writeln('Get (name) : ',GetInt64Prop(O,'Int64Field'));
  Writeln('Get (propinfo) : ',GetInt64Prop(O,PI));
  SetInt64Prop(O,'Int64Field',12345);
  Writeln('Set (name,12345) : ',O.Int64Field);
  SetInt64Prop(O,PI,54321);
  Writeln('Set (propinfo,54321) : ',O.Int64Field);
  O.Free;
end.

```

78.7.15 GetInterfaceProp

Synopsis: Return interface-typed property.

Declaration:

```

function GetInterfaceProp(Instance: TObject; const PropName: string)
    : IInterface
function GetInterfaceProp(Instance: TObject; PropInfo: PPropInfo)
    : IInterface

```

Visibility: default

Description: GetInterfaceProp returns the interface which the property described by PropInfo or with name Propname points to for object Instance.

Errors: No checking is done whether Instance is non-nil, or whether PropInfo describes a valid method property of Instance. Specifying an invalid property name in PropName will result in an EPropertyError exception.

See also: SetInterfaceProp (2106), GetOrdProp (2092), GetStrProp (2098), GetFloatProp (2086), GetInt64Prop (2087), GetSetProp (2097), GetObjectProp (2090), GetEnumProp (2085)

78.7.16 GetMethodProp

Synopsis: Return value of a method property.

Declaration:

```

function GetMethodProp(Instance: TObject; PropInfo: PPropInfo) : TMethod
function GetMethodProp(Instance: TObject; const PropName: string)
    : TMethod

```

Visibility: default

Description: `GetMethodProp` returns the method the property described by `PropInfo` or with name `Propname` for object `Instance`. The return type `TMethod` is defined in the `SysUtils` unit as:

```
TMethod = packed record
  Code, Data: Pointer;
end;
```

`Data` points to the instance of the class with the method `Code`.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid method property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: `SetMethodProp` (2106), `GetOrdProp` (2092), `GetStrProp` (2098), `GetFloatProp` (2086), `GetInt64Prop` (2087), `GetSetProp` (2097), `GetObjectProp` (2090), `GetEnumProp` (2085)

Listing: `./examples/typinfex/ex6.pp`

program example6;

{ This program demonstrates the GetMethodProp function }

{ \$mode objfpc }

uses rttiobj, typinfo, sysutils;

Type

```
TNotifyObject = Class(TObject)
  Procedure Notification1(Sender : TObject);
  Procedure Notification2(Sender : TObject);
end;
```

Procedure TNotifyObject.Notification1(Sender : TObject);

begin

```
  Write('Received notification 1 of object with class: ');
  Writeln(Sender.ClassName);
```

end;

Procedure TNotifyObject.Notification2(Sender : TObject);

begin

```
  Write('Received notification 2 of object with class: ');
  Writeln(Sender.ClassName);
```

end;

Var

```
O : TMyTestObject;
PI : PPropInfo;
NO : TNotifyObject;
M : TMethod;
```

Procedure PrintMethod (**Const** M : TMethod);

begin

```
  If (M.Data=Pointer(NO)) Then
```

```

If (M.Code=Pointer(@TNotifyObject.Notification1)) then
  Writeln('Notification1')
else If (M.Code=Pointer(@TNotifyObject.Notification2)) then
  Writeln('Notification2')
else
  begin
    Write('Unknown method adress (data:');
    Write(hexStr(Longint(M.data),8));
    Writeln(',code:',hexstr(Longint(M.Code),8),');
  end;
end;

```

```

begin
  O:=TMyTestObject.Create;
  NO:=TNotifyObject.Create;
  O.NotifyEvent:=@NO.Notification1;
  PI:=GetPropInfo(O,'NotifyEvent');
  Writeln('Method property : ');
  Write('Notifying : ');
  O.Notify;
  Write('Get (name) : ');
  M:=GetMethodProp(O,'NotifyEvent');
  PrintMethod(M);
  Write('Notifying : ');
  O.Notify;
  Write('Get (propinfo) : ');
  M:=GetMethodProp(O,PI);
  PrintMethod(M);
  M:=TMethod(@NO.Notification2);
  SetMethodProp(O,'NotifyEvent',M);
  Write('Set (name,Notification2) : ');
  M:=GetMethodProp(O,PI);
  PrintMethod(M);
  Write('Notifying : ');
  O.Notify;
  Write('Set (propinfo,Notification1) : ');
  M:=TMethod(@NO.Notification1);
  SetMethodProp(O,PI,M);
  M:=GetMethodProp(O,PI);
  PrintMethod(M);
  Write('Notifying : ');
  O.Notify;
  O.Free;
end.

```

78.7.17 TObjectProp

Synopsis: Return value of an object-type property.

```

Declaration: function TObjectProp(Instance: TObject; const PropName: string)
                    : TObject
function TObjectProp(Instance: TObject; const PropName: string;
                    MinClass: TClass) : TObject
function TObjectProp(Instance: TObject; PropInfo: PPropInfo) : TObject
function TObjectProp(Instance: TObject; PropInfo: PPropInfo;
                    MinClass: TClass) : TObject

```

Visibility: default

Description: `GetObjectProp` returns the object which the property described by `PropInfo` with name `Propname` points to for object `Instance`.

If `MinClass` is specified, then if the object is not descendent of class `MinClass`, then `Nil` is returned.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid method property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: `SetMethodProp` (2106), `GetOrdProp` (2092), `GetStrProp` (2098), `GetFloatProp` (2086), `GetInt64Prop` (2087), `GetSetProp` (2097), `GetObjectProp` (2090), `GetEnumProp` (2085)

Listing: `./examples/typinfex/ex5.pp`

program example5;

{ This program demonstrates the GetObjectProp function }

{ \$mode objfpc }

uses rttiobj, typinfo;

Var

O : TMyTestObject;
PI : PPropInfo;
NO1, NO2 : TNamedObject;

begin

```
O:=TMyTestObject.Create;
NO1:=TNamedObject.Create;
NO1.ObjectName:='First named object';
NO2:=TNamedObject.Create;
NO2.ObjectName:='Second named object';
O.ObjField:=NO1;
Writeln('Object property : ');
PI:=GetPropInfo(O,'ObjField');
Write('Property class : ');
Writeln(GetObjectPropClass(O,'ObjField').ClassName);
Write('Value : ');
Writeln((O.ObjField as TNamedObject).ObjectName);
Write('Get (name) : ');
Writeln((GetObjectProp(O,'ObjField') as TNamedObject).ObjectName);
Write('Get (propinfo) : ');
Writeln((GetObjectProp(O,PI,TObject) as TNamedObject).ObjectName);
SetObjectProp(O,'ObjField',NO2);
Write('Set (name,NO2) : ');
Writeln((O.ObjField as TNamedObject).ObjectName);
SetObjectProp(O,PI,NO1);
Write('Set (propinfo,NO1) : ');
Writeln((O.ObjField as TNamedObject).ObjectName);
O.Free;
end.
```

78.7.18 GetObjectPropClass

Synopsis: Return class of property.

Declaration: `function GetObjectPropClass(Instance: TObject; const PropName: string)
: TClass
function GetObjectPropClass(AClass: TClass; const PropName: string)
: TClass`

Visibility: default

Description: `GetObjectPropClass` returns the declared class of the property with name `PropName`. This may not be the actual class of the property value.

For an example, see `GetObjectProp` (2090).

Errors: No checking is done whether `Instance` is non-nil. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: `SetMethodProp` (2106), `GetOrdProp` (2092), `GetStrProp` (2098), `GetFloatProp` (2086), `GetInt64Prop` (2087)

78.7.19 GetOrdProp

Synopsis: Get the value of an ordinal property.

Declaration: `function GetOrdProp(Instance: TObject; PropInfo: PPropInfo) : Int64
function GetOrdProp(Instance: TObject; const PropName: string) : Int64`

Visibility: default

Description: `GetOrdProp` returns the value of the ordinal property described by `PropInfo` or with name `PropName` for the object `Instance`. The value is returned as a longint, which should be typecast to the needed type.

Ordinal properties that can be retrieved include:

Integers and subranges of integers The value of the integer will be returned.

Enumerated types and subranges of enumerated types The ordinal value of the enumerated type will be returned.

Sets If the base type of the set has less than 31 possible values. If a bit is set in the return value, then the corresponding element of the base ordinal class of the set type must be included in the set.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid ordinal property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: `SetOrdProp` (2107), `GetStrProp` (2098), `GetFloatProp` (2086), `GetInt64Prop` (2087), `GetMethodProp` (2088), `GetSetProp` (2097), `GetObjectProp` (2090), `GetEnumProp` (2085)

Listing: `./examples/typinfex/ex1.pp`

```
program example1;

{ This program demonstrates the GetOrdProp function }

{$mode objfpc}

uses rttiobj, typinfo;

Var
  O : TMyTestObject;
```

```

PI : PPropInfo;

begin
  O:=TMyTestObject.Create;
  Writeln('Boolean property : ');
  Writeln('Value : ',O.BooleanField);
  Writeln('Ord(Value) : ',Ord(O.BooleanField));
  Writeln('Get (name) : ',GetOrdProp(O,'BooleanField'));
  PI:=GetPropInfo(O,'BooleanField');
  Writeln('Get (propinfo) : ',GetOrdProp(O,PI));
  SetOrdProp(O,'BooleanField',Ord(False));
  Writeln('Set (name,false) : ',O.BooleanField);
  SetOrdProp(O,PI,Ord(True));
  Writeln('Set (propinfo,true) : ',O.BooleanField);
  O.Free;
end.

```

78.7.20 GetPropInfo

Synopsis: Return property type information, by property name.

Declaration: `function GetPropInfo (TypeInfo: PTypeInfo; const PropName: string) : PPropInfo`
`function GetPropInfo (TypeInfo: PTypeInfo; const PropName: string; AKinds: TTypeKinds) : PPropInfo`
`function GetPropInfo (Instance: TObject; const PropName: string) : PPropInfo`
`function GetPropInfo (Instance: TObject; const PropName: string; AKinds: TTypeKinds) : PPropInfo`
`function GetPropInfo (AClass: TClass; const PropName: string) : PPropInfo`
`function GetPropInfo (AClass: TClass; const PropName: string; AKinds: TTypeKinds) : PPropInfo`

Visibility: default

Description: `GetPropInfo` returns a pointer to the `TPropInfo` record for the `PropName` property of a class. The class to examine can be specified in one of three ways:

Instance An instance of the class.

AClass A class pointer to the class.

TypeInfo A pointer to the type information of the class.

In each of these three ways, if `AKinds` is specified, if the property has `TypeKind` which is not included in `AKinds`, `Nil` will be returned.

For an example, see most of the other functions.

Errors: If the property `PropName` does not exist, `Nil` is returned.

See also: `GetPropInfos` ([2093](#)), `GetPropList` ([2094](#))

78.7.21 GetPropInfos

Synopsis: Return a list of published properties.

Declaration: `procedure GetPropInfos (TypeInfo: PTypeInfo; PropList: PPropList)`

Visibility: default

Description: `GetPropInfos` stores pointers to the property information of all published properties of a class with class info `TypeInfo` in the list pointed to by `PropList`. The `PropList` pointer must point to a memory location that contains enough space to hold all properties of the class and its parent classes.

Errors: No checks are done to see whether `PropList` points to a memory area that is big enough to hold all pointers.

See also: `GetPropInfo` (2093), `GetPropList` (2094)

Listing: ./examples/typinfex/ex12.pp

Program example12;

{ This program demonstrates the GetPropInfos function }

uses

rttiobj, typinfo;

Var

O : TMyTestObject;

PT : PTypeData;

PI : PTypeInfo;

I,J : Longint;

PP : PPropList;

prl : PPropInfo;

begin

O:=TMyTestObject.Create;

PI:=O.ClassInfo;

PT:=GetTypeData(PI);

WriteLn('Property Count : ',PT^.PropCount);

GetMem (PP,PT^.PropCount*SizeOf(Pointer));

GetPropInfos(PI,PP);

For I:=0 to PT^.PropCount-1 do

begin

With PP^[i]^ do

begin

Write('Property ',i+1:3,' : ',name:30);

writeln(' Type: ',TypeNames[typinfo.PropType(O,Name)]);

end;

end;

FreeMem(PP);

O.Free;

end.

78.7.22 GetPropList

Synopsis: Return a list of a certain type of published properties.

Declaration:

```
function GetPropList (TypeInfo: PTypeInfo; TypeKinds: TTypeKinds;
                     PropList: PPropList; Sorted: Boolean) : LongInt
function GetPropList (TypeInfo: PTypeInfo; out PropList: PPropList)
                     : SizeInt
```

```
function GetPropList (AClass: TClass; out PropList: PPropList) : Integer
function GetPropList (Instance: TObject; out PropList: PPropList)
    : Integer
```

Visibility: default

Description: `GetPropList` stores pointers to property information of the class with class info `TypeInfo` for properties of kind `TypeKinds` in the list pointed to by `PropList`. `PropList` must contain enough space to hold all properties.

The function returns the number of pointers that matched the criteria and were stored in `PropList`.

Errors: No checks are done to see whether `PropList` points to a memory area that is big enough to hold all pointers.

See also: `GetPropInfos` ([2093](#)), `GetPropInfo` ([2093](#))

Listing: `./examples/typinfex/ex13.pp`

Program `example13;`

{ This program demonstrates the GetPropList function }

uses

`rttiobj,typinfo;`

Var

`O : TMyTestObject;`

`PT : PTypeData;`

PI : PTypeInfo;

`I,J : Longint;`

`PP : PPropList;`

`prl : PPropInfo;`

begin

`O:=TMyTestObject.Create;`

PI:=O.ClassInfo;

`PT:=GetTypeData(PI);`

WriteLn("Total property Count : ",PT^.PropCount);

GetMem (PP,PT^.PropCount***SizeOf**(Pointer));

`J:=GetPropList(PI,OrdinalTypes,PP);`

WriteLn('Ordinal property Count : ',J);

For I:=0 to J-1 **do**

begin

With PP^[i]^ **do**

begin

Write('Property ',i+1:3,' : ',name:30);

writeln(' Type: ',TypeNames[typinfo.PropType(O,Name)]);

end;

end;

FreeMem(PP);

`O.Free;`

end.

78.7.23 GetPropValue

Synopsis: Get property value as a variant.

Declaration:

```
function GetPropValue(Instance: TObject; const PropName: string)
    : Variant
function GetPropValue(Instance: TObject; const PropName: string;
    PreferStrings: Boolean) : Variant
function GetPropValue(Instance: TObject; PropInfo: PPropInfo) : Variant
function GetPropValue(Instance: TObject; PropInfo: PPropInfo;
    PreferStrings: Boolean) : Variant
```

Visibility: default

Description: GetPropValue returns the value of a property of the object Instance as a Variant value. The property can be specified as a name using PropName, or using property RTTI info in PropInfo. If PreferStrings is True then a string value is returned where possible.

The Variants unit must be added to your project for this function to be able to do its work properly.

See also: GetPropInfos ([2093](#)), GetPropInfo ([2093](#)), SetPropValue ([2108](#))

78.7.24 GetRawByteStrProp

Synopsis: Return the value of a string property as a RawByteString.

Declaration:

```
function GetRawByteStrProp(Instance: TObject; PropInfo: PPropInfo)
    : RawByteString
function GetRawByteStrProp(Instance: TObject; const PropName: string)
    : RawByteString
```

Visibility: default

Description: GetRawByteStrProp returns the value of a string property of the object Instance as a RawByteString. This means it has the codepage of the actual value, no codepage translation is done. The property to read can be specified using the name PropName or the property info PropInfo.

See also: GetStrProp ([2098](#))

78.7.25 GetRawInterfaceProp

Synopsis: Get a raw (CORBA) interface property.

Declaration:

```
function GetRawInterfaceProp(Instance: TObject; const PropName: string)
    : Pointer
function GetRawInterfaceProp(Instance: TObject; PropInfo: PPropInfo)
    : Pointer
```

Visibility: default

Description: GetRawInterfaceProp can be used to retrieve the value of a published CORBA interface property with name PropName from object Instance. Alternatively, the required property information can be specified by PropInfo instead of the property name. In difference with the GetInterfaceProp ([2088](#)) function, no reference counting is done.

Errors: If the property PropName does not exist, an EPropertyError exception is raised.

See also: GetInterfaceProp ([2088](#)), SetRawInterfaceProp ([2108](#))

78.7.26 GetSetProp

Synopsis: Return the value of a set property.

Declaration:

```
function GetSetProp(Instance: TObject; const PropName: string) : string
function GetSetProp(Instance: TObject; const PropName: string;
                    Brackets: Boolean) : string
function GetSetProp(Instance: TObject; const PropInfo: PPropInfo;
                    Brackets: Boolean) : string
```

Visibility: default

Description: `GetSetProp` returns the contents of a set property as a string. The property to be returned can be specified by its name in `PropName` or by its property information in `PropInfo`.

The returned set is a string representation of the elements in the set as returned by `SetToString` (2109). The `Brackets` option can be used to enclose the string representation in square brackets.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid ordinal property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: `SetSetProp` (2109), `GetStrProp` (2098), `GetFloatProp` (2086), `GetInt64Prop` (2087), `GetMethodProp` (2088)

Listing: ./examples/typinfex/ex7.pp

```
program example7;

{ This program demonstrates the GetSetProp function }

{$mode objfpc}

uses rttiobj, typinfo;

Var
  O : TMyTestObject;
  PI : PPropInfo;

Function SetAsString (ASet : TMyEnums) : String;

Var
  i : TmyEnum;

begin
  result:="";
  For i:=mefirst to methird do
    If i in ASet then
      begin
        If (Result<>") then
          Result:=Result+' ';
        Result:=Result+MyEnumNames[i];
      end;
  end;

Var
  S : TMyEnums;

begin
  O:=TMyTestObject.Create;
```

```

O.SetField:=[meFirst,meSecond,meThird];
Writeln('Set property : ');
Writeln('Value : ',SetAsString(O.SetField));
Writeln('Ord(Value) : ',Byte(O.SetField));
Writeln('Get (name) : ',GetSetProp(O,'SetField'));
PI:=GetPropInfo(O,'SetField');
Writeln('Get (propinfo) : ',GetSetProp(O,PI,false));
S:=[meFirst,meThird];
SetOrdProp(O,'SetField',Byte(S));
Write('Set (name,[meFirst,meThird]) : ');
Writeln(SetAsString(O.SetField));
S:=[meSecond];
SetOrdProp(O,PI,Byte(S));
Write('Set (propinfo,[meSecond]) : ');
Writeln(SetAsString(O.SetField));
O.Free;
end.

```

78.7.27 GetStrProp

Synopsis: Return the value of a string property.

Declaration: `function GetStrProp(Instance: TObject; PropInfo: PPropInfo) : Ansistring`
`function GetStrProp(Instance: TObject; const PropName: string) : string`

Visibility: default

Description: `GetStrProp` returns the value of the string property described by `PropInfo` or with name `PropName` for object `Instance`.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid string property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: `SetStrProp` (2109), `SetWideStrProp` (2111), `GetOrdProp` (2092), `GetFloatProp` (2086), `GetInt64Prop` (2087), `GetMethodProp` (2088)

Listing: `./examples/typinfex/ex3.pp`

```

program example3;

{ This program demonstrates the GetStrProp function }

{$mode objfpc}

uses rttiobj,typinfo;

Var
  O : TMyTestObject;
  PI : PPropInfo;

begin
  O:=TMyTestObject.Create;
  PI:=GetPropInfo(O,'AnsiStringField');
  Writeln('String property : ');
  Writeln('Value : ',O.AnsiStringField);
  Writeln('Get (name) : ',GetStrProp(O,'AnsiStringField'));

```


78.7.31 GetWideStrProp

Synopsis: Read a widestring property.

```
Declaration: function GetWideStrProp(Instance: TObject; PropInfo: PPropInfo)
              : WideString
function GetWideStrProp(Instance: TObject; const PropName: string)
              : WideString
```

Visibility: default

Description: `GetWideStrProp` returns the value of the widestring property described by `PropInfo` or with name `PropName` for object `Instance`.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid widestring property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: [GetStrProp \(2098\)](#), [SetWideStrProp \(2111\)](#), [GetOrdProp \(2092\)](#), [GetFloatProp \(2086\)](#), [GetInt64Prop \(2087\)](#), [GetMethodProp \(2088\)](#)

78.7.32 IsPublishedProp

Synopsis: Check whether a published property exists.

```
Declaration: function IsPublishedProp(Instance: TObject; const PropName: string)
              : Boolean
function IsPublishedProp(AClass: TClass; const PropName: string)
              : Boolean
```

Visibility: default

Description: `IsPublishedProp` returns true if a class has a published property with name `PropName`. The class can be specified in one of two ways:

AClass A class pointer to the class.

Instance An instance of the class.

Errors: No checks are done to ensure `Instance` or `AClass` are valid pointers. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: [IsStoredProp \(2101\)](#), [PropIsType \(2102\)](#)

Listing: ./examples/typinfex/ex10.pp

```
program example10;

{ This program demonstrates the IsPublishedProp function }

{$mode objfpc}

uses rttiobj, typinfo;

Var
    O : TMyTestObject;
    PI : PPropInfo;

begin
```

```

program example11;

{ This program demonstrates the IsStoredProp function }

{$mode objfpc}

uses rttiobj,typinfo;

Var
  O : TMyTestObject;
  PI : PPropInfo;

begin
  O:=TMyTestObject.Create;
  WriteLn('Stored tests : ');
  Write('IsStoredProp(O,StoredIntegerConstFalse) : ');
  WriteLn(IsStoredProp(O,'StoredIntegerConstFalse'));
  Write('IsStoredProp(O,StoredIntegerConstTrue) : ');
  WriteLn(IsStoredProp(O,'StoredIntegerConstTrue'));
  Write('IsStoredProp(O,StoredIntegerMethod) : ');
  WriteLn(IsStoredProp(O,'StoredIntegerMethod'));
  Write('IsStoredProp(O,StoredIntegerVirtualMethod) : ');
  WriteLn(IsStoredProp(O,'StoredIntegerVirtualMethod'));
  O.Free;
end.

```

78.7.35 IsWriteableProp

Synopsis: Can the property be written.

Declaration:

```

function IsWriteableProp(PropInfo: PPropInfo) : Boolean
function IsWriteableProp(Instance: TObject; const PropName: string)
    : Boolean
function IsWriteableProp(AClass: TClass; const PropName: string)
    : Boolean

```

Visibility: default

Description: IsWriteableProp returns True if the property described by information PropInfo can be written to, i.e. has a write specifier. It returns False if the property can only be read.

See also: IsReadableProp ([2101](#))

78.7.36 PropIsType

Synopsis: Check the type of a published property.

Declaration:

```

function PropIsType(Instance: TObject; const PropName: string;
    TypeKind: TTypeKind) : Boolean
function PropIsType(AClass: TClass; const PropName: string;
    TypeKind: TTypeKind) : Boolean

```

Visibility: default

Description: PropIsType returns True if the property with name PropName has type TypeKind. It returns False otherwise. The class to be examined can be specified in one of two ways:

AClass A class pointer.

Instance An instance of the class.

Errors: No checks are done to ensure `Instance` or `AClass` are valid pointers. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: `IsPublishedProp` (2100), `IsStoredProp` (2101), `PropType` (2103)

Listing: `./examples/typinfex/ex16.pp`

```

program example16;

{ This program demonstrates the PropIsType function }

{$mode objfpc}

uses rttiobj, typinfo;

Var
  O : TMyTestObject;

begin
  O := TMyTestObject.Create;
  Writeln('Property tests : ');
  Write('PropIsType(O, BooleanField, tkBool) : ');
  Writeln(PropIsType(O, 'BooleanField', tkBool));
  Write('PropIsType(Class, BooleanField, tkBool) : ');
  Writeln(PropIsType(O.ClassType, 'BooleanField', tkBool));
  Write('PropIsType(O, ByteField, tkString) : ');
  Writeln(PropIsType(O, 'ByteField', tkString));
  Write('PropIsType(Class, ByteField, tkString) : ');
  Writeln(PropIsType(O.ClassType, 'ByteField', tkString));
  O.Free;
end.

```

78.7.37 PropType

Synopsis: Return the type of a property.

Declaration: `function PropType(Instance: TObject; const PropName: string) : TTypeKind`
`function PropType(AClass: TClass; const PropName: string) : TTypeKind`

Visibility: default

Description: `PropType` returns the type of the property `PropName` for a class. The class to be examined can be specified in one of 2 ways:

AClass A class pointer.

Instance An instance of the class.

Errors: No checks are done to ensure `Instance` or `AClass` are valid pointers. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: `IsPublishedProp` (2100), `IsStoredProp` (2101), `PropIsType` (2102)

Listing: `./examples/typinfex/ex17.pp`

```

program example17;

{ This program demonstrates the PropType function }

{$mode objfpc}

uses rttiobj, typinfo;

Var
  O : TMyTestObject;

begin
  O:=TMyTestObject.Create;
  Writeln('Property tests : ');
  Write('PropType(O,BooleanField) : ');
  Writeln(TypeNames[PropType(O,'BooleanField')]);
  Write('PropType(Class,BooleanField) : ');
  Writeln(TypeNames[PropType(O.ClassType,'BooleanField')]);
  Write('PropType(O,ByteField) : ');
  Writeln(TypeNames[PropType(O,'ByteField')]);
  Write('PropType(Class,ByteField) : ');
  Writeln(TypeNames[PropType(O.ClassType,'ByteField')]);
  O.Free;
end.

```

78.7.38 RemoveEnumElementAliases

Synopsis: Remove aliases for enumerated elements.

Declaration: `procedure RemoveEnumElementAliases (aTypeInfo: PTypeInfo)`

Visibility: default

Description: `RemoveEnumElementAliases` removes all aliases for enumerated type `aTypeInfo`. The aliases must have been defined using `AddEnumElementAliases` (2081).

Errors: `AddEnumElementAliases` (2081) `GetEnumValue` (2086) `GetEnumeratedAliasValue` (2084)

78.7.39 SetDynArrayProp

Synopsis: `SetSize` the value of a dynamic array valued property.

Declaration: `procedure SetDynArrayProp (Instance: TObject; const PropName: string; const Value: Pointer)`
`procedure SetDynArrayProp (Instance: TObject; PropInfo: PPropInfo; const Value: Pointer)`

Visibility: default

Description: `GetDynArrayProp` returns the value of a dynamic array valued property of the object `Instance` to the pointer `Value`. The property to read can be specified using the name `PropName` or the property info `PropInfo`.

See also: `GetDynArrayProp` (2083)

78.7.40 SetEnumProp

Synopsis: Set value of an enumerated-type property.

Declaration:

```
procedure SetEnumProp(Instance: TObject; const PropName: string;
                      const Value: string)
procedure SetEnumProp(Instance: TObject; const PropInfo: PPropInfo;
                      const Value: string)
```

Visibility: default

Description: `SetEnumProp` sets the property described by `PropInfo` or with name `PropName` to `Value`. `Value` must be a string with the name of the enumerate value, i.e. it can be used as an argument to `GetEnumValue` (2086).

For an example, see `GetEnumProp` (2085).

Errors: No checks are done to ensure `Instance` or `PropInfo` are valid pointers. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: `GetEnumProp` (2085), `SetStrProp` (2109), `SetFloatProp` (2105), `SetInt64Prop` (2105), `SetMethodProp` (2106)

78.7.41 SetFloatProp

Synopsis: Set value of a float property.

Declaration:

```
procedure SetFloatProp(Instance: TObject; const PropName: string;
                      Value: Extended)
procedure SetFloatProp(Instance: TObject; PropInfo: PPropInfo;
                      Value: Extended)
```

Visibility: default

Description: `SetFloatProp` assigns `Value` to the property described by `PropInfo` or with name `Propname` for the object `Instance`.

For an example, see `GetFloatProp` (2086).

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid float property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: `GetFloatProp` (2086), `SetOrdProp` (2107), `SetStrProp` (2109), `SetInt64Prop` (2105), `SetMethodProp` (2106)

78.7.42 SetInt64Prop

Synopsis: Set value of a `Int64` property.

Declaration:

```
procedure SetInt64Prop(Instance: TObject; PropInfo: PPropInfo;
                      const Value: Int64)
procedure SetInt64Prop(Instance: TObject; const PropName: string;
                      const Value: Int64)
```

Visibility: default

Description: `SetInt64Prop` assigns `Value` to the property of type `Int64` that is described by `PropInfo` or with name `Propname` for the object `Instance`.

For an example, see `GetInt64Prop` (2087).

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid `Int64` property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: [GetInt64Prop \(2087\)](#), [GetMethodProp \(2088\)](#), [SetOrdProp \(2107\)](#), [SetStrProp \(2109\)](#), [SetFloatProp \(2105\)](#)

78.7.43 SetInterfaceProp

Synopsis: Set interface-valued property.

Declaration:

```
procedure SetInterfaceProp(Instance: TObject; const PropName: string;
                           const Value: IInterface)
procedure SetInterfaceProp(Instance: TObject; PropInfo: PPropInfo;
                           const Value: IInterface)
```

Visibility: default

Description: `SetInterfaceProp` assigns `Value` to the object property described by `PropInfo` or with name `Propname` for the object `Instance`.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid interface property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: [GetInterfaceProp \(2088\)](#), [SetObjectProp \(2107\)](#), [SetOrdProp \(2107\)](#), [SetStrProp \(2109\)](#), [SetFloatProp \(2105\)](#), [SetInt64Prop \(2105\)](#), [SetMethodProp \(2106\)](#)

78.7.44 SetMethodProp

Synopsis: Set the value of a method property.

Declaration:

```
procedure SetMethodProp(Instance: TObject; PropInfo: PPropInfo;
                        const Value: TMethod)
procedure SetMethodProp(Instance: TObject; const PropName: string;
                        const Value: TMethod)
```

Visibility: default

Description: `SetMethodProp` assigns `Value` to the method the property described by `PropInfo` or with name `Propname` for object `Instance`.

The type `TMethod` of the `Value` parameter is defined in the `SysUtils` unit as:

```
TMethod = packed record
    Code, Data: Pointer;
end;
```

`Data` should point to the instance of the class with the method `Code`.

For an example, see [GetMethodProp \(2088\)](#).

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid method property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: [GetMethodProp \(2088\)](#), [SetOrdProp \(2107\)](#), [SetStrProp \(2109\)](#), [SetFloatProp \(2105\)](#), [SetInt64Prop \(2105\)](#)

78.7.45 SetObjectProp

Synopsis: Set the value of an object-type property.

Declaration:

```
procedure SetObjectProp(Instance: TObject; const PropName: string;
                        Value: TObject)
procedure SetObjectProp(Instance: TObject; PropInfo: PPropInfo;
                        Value: TObject)
```

Visibility: default

Description: `SetObjectProp` assigns `Value` to the object property described by `PropInfo` or with name `Propname` for the object `Instance`.

For an example, see `GetObjectProp` (2090).

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid object property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: `GetObjectProp` (2090), `SetOrdProp` (2107), `SetStrProp` (2109), `SetFloatProp` (2105), `SetInt64Prop` (2105), `SetMethodProp` (2106)

78.7.46 SetOrdProp

Synopsis: Set value of an ordinal property.

Declaration:

```
procedure SetOrdProp(Instance: TObject; PropInfo: PPropInfo;
                    Value: Int64)
procedure SetOrdProp(Instance: TObject; const PropName: string;
                    Value: Int64)
```

Visibility: default

Description: `SetOrdProp` assigns `Value` to the ordinal property described by `PropInfo` or with name `Propname` for the object `Instance`.

Ordinal properties that can be set include:

Integers and subranges of integers The actual value of the integer must be passed.

Enumerated types and subranges of enumerated types The ordinal value of the enumerated type must be passed.

Subrange types of integers or enumerated types. Here the ordinal value must be passed.

Sets If the base type of the set has less than 31 possible values. For each possible value; the corresponding bit of `Value` must be set.

For an example, see `GetOrdProp` (2092).

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid ordinal property of `Instance`. No range checking is performed. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: `GetOrdProp` (2092), `SetStrProp` (2109), `SetFloatProp` (2105), `SetInt64Prop` (2105), `SetMethodProp` (2106)

78.7.47 SetPropValue

Synopsis: Set property value as variant.

Declaration:

```
procedure SetPropValue(Instance: TObject; const PropName: string;
                      const Value: Variant)
procedure SetPropValue(Instance: TObject; PropInfo: PPropInfo;
                      const Value: Variant)
```

Visibility: default

Description: `SetPropValue` sets the value of a property of the object `Instance` as a `Variant` value, specified in `Value`. The property can be specified as a name using `PropName`, or using property RTTI info in `PropInfo`.

The `Variants` unit must be added to your project for this function to be able to do its work properly.

See also: `GetPropValue` ([2095](#))

78.7.48 SetRawByteStrProp

Synopsis: Set the value of a string property using a `RawByteString`.

Declaration:

```
procedure SetRawByteStrProp(Instance: TObject; const PropName: string;
                          const Value: RawByteString)
procedure SetRawByteStrProp(Instance: TObject; PropInfo: PPropInfo;
                          const Value: RawByteString)
```

Visibility: default

Description: `GetRawByteStrProp` writes the value of a string property of the object `Instance` to `Value`, a `RawByteString`. This means no codepage translation is done. The property to write can be specified using the name `PropName` or the property info `PropInfo`.

See also: `SetStrProp` ([2109](#)), `GetRawByteStrProp` ([2096](#))

78.7.49 SetRawInterfaceProp

Synopsis: Set a raw (CORBA) interface property.

Declaration:

```
procedure SetRawInterfaceProp(Instance: TObject;
                             const PropName: string;
                             const Value: Pointer)
procedure SetRawInterfaceProp(Instance: TObject; PropInfo: PPropInfo;
                             const Value: Pointer)
```

Visibility: default

Description: `SetRawInterfaceProp` can be used to set the value of a published CORBA interface with name `PropName` from object `Instance` to `Value`. Alternatively, the required property information can be specified by `PropInfo` instead of the property name. In difference with the `SetInterfaceProp` ([2106](#)) procedure, no reference counting is done.

Errors: If the property `PropName` does not exist, an `EPropertyError` exception is raised.

See also: `SetInterfaceProp` ([2106](#)), `GetRawInterfaceProp` ([2096](#))

78.7.50 SetSetProp

Synopsis: Set value of set-typed property.

Declaration: `procedure SetSetProp(Instance: TObject; const PropName: string;
const Value: string)
procedure SetSetProp(Instance: TObject; const PropInfo: PPropInfo;
const Value: string)`

Visibility: default

Description: SetSetProp sets the property specified by PropInfo or PropName for object Instance to Value. Value is a string which contains a comma-separated list of values, each value being a string-representation of the enumerated value that should be included in the set. The value should be accepted by the StringToSet (2111) function.

The value can be formed using the SetToString (2109) function.

For an example, see GetSetProp (2097).

Errors: No checking is done whether Instance is non-nil, or whether PropInfo describes a valid ordinal property of Instance. No range checking is performed. Specifying an invalid property name in PropName will result in an EPropertyError exception.

See also: GetSetProp (2097), SetOrdProp (2107), SetStrProp (2109), SetFloatProp (2105), SetInt64Prop (2105), SetMethodProp (2106), SetToString (2109), StringToSet (2111)

78.7.51 SetStrProp

Synopsis: Set value of a string property.

Declaration: `procedure SetStrProp(Instance: TObject; const PropName: string;
const Value: AnsiString)
procedure SetStrProp(Instance: TObject; PropInfo: PPropInfo;
const Value: Ansistring)`

Visibility: default

Description: SetStrProp assigns Value to the string property described by PropInfo or with name Propname for object Instance.

For an example, see GetStrProp (2098)

Errors: No checking is done whether Instance is non-nil, or whether PropInfo describes a valid string property of Instance. Specifying an invalid property name in PropName will result in an EPropertyError exception.

See also: GetStrProp (2098), SetWideStrProp (2111), SetOrdProp (2107), SetFloatProp (2105), SetInt64Prop (2105), SetMethodProp (2106)

78.7.52 SetToString

Synopsis: Convert set to a string description.

Declaration: `function SetToString(TypeInfo: PTypeInfo; Value: LongInt;
Brackets: Boolean) : string
function SetToString(PropInfo: PPropInfo; Value: LongInt;
Brackets: Boolean) : string
function SetToString(PropInfo: PPropInfo; Value: LongInt) : string`

```

function SetToString(TypeInfo: PTypeInfo; Value: Pointer;
                    Brackets: Boolean) : string
function SetToString(PropInfo: PPropInfo; Value: Pointer;
                    Brackets: Boolean) : string

```

Visibility: default

Description: `SetToString` takes an integer representation of a set (as received e.g. by `GetOrdProp`) and turns it into a string representing the elements in the set, based on the type information found in the `PropInfo` property information. By default, the string representation is not surrounded by square brackets. Setting the `Brackets` parameter to `True` will surround the string representation with brackets.

The function returns the string representation of the set.

Errors: No checking is done to see whether `PropInfo` points to valid property information.

See also: `GetEnumName` (2084), `GetEnumValue` (2086), `StringToSet` (2111)

Listing: `./examples/typinfex/ex18.pp`

```

program example18;

{ This program demonstrates the SetToString function }

{$mode objfpc}

uses rttiobj, typinfo;

Var
  O : TMyTestObject;
  PI : PPropInfo;
  I : longint;

begin
  O := TMyTestObject.Create;
  PI := GetPropInfo(O, 'SetField');
  O.SetField := [meFirst, meSecond, meThird];
  I := GetOrdProp(O, PI);
  Writeln('Set property to string : ');
  Writeln('Value : ', SetToString(PI, I, False));
  O.SetField := [meFirst, meSecond];
  I := GetOrdProp(O, PI);
  Writeln('Value : ', SetToString(PI, I, True));
  I := StringToSet(PI, 'meFirst');
  SetOrdProp(O, PI, I);
  I := GetOrdProp(O, PI);
  Writeln('Value : ', SetToString(PI, I, False));
  I := StringToSet(PI, [meSecond, meThird]);
  SetOrdProp(O, PI, I);
  I := GetOrdProp(O, PI);
  Writeln('Value : ', SetToString(PI, I, True));
  O.Free;
end.

```

78.7.53 SetUnicodeStrProp

Synopsis: Set UnicodeString-valued property.

Declaration: `procedure SetUnicodeStrProp(Instance: TObject; const PropName: string;
const Value: UnicodeString)
procedure SetUnicodeStrProp(Instance: TObject; PropInfo: PPropInfo;
const Value: UnicodeString)`

Visibility: default

Description: `SetUnicodeStrProp` sets the `UnicodeString` property from `Instance` to `Value`, where the property is identified by the `PropInfo` pointer or the `PropertyName`.

Errors: If no property of the indicated name exists, or it is not of type `UnicodeString`, an exception will occur.

See also: `SetStrProp` (2109), `GetUnicodeStrProp` (2099)

78.7.54 SetVariantProp

Synopsis: Set value of a variant property.

Declaration: `procedure SetVariantProp(Instance: TObject; const PropName: string;
const Value: Variant)
procedure SetVariantProp(Instance: TObject; PropInfo: PPropInfo;
const Value: Variant)`

Visibility: default

Description: Due to missing `Variant` support, this function is not yet implemented. Provided for Delphi compatibility only.

78.7.55 SetWideStrProp

Synopsis: Set a widestring property.

Declaration: `procedure SetWideStrProp(Instance: TObject; const PropName: string;
const Value: WideString)
procedure SetWideStrProp(Instance: TObject; PropInfo: PPropInfo;
const Value: WideString)`

Visibility: default

Description: `SetWideStrProp` assigns `Value` to the widestring property described by `PropInfo` or with name `Propname` for object `Instance`.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid widestring property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: `GetWideStrProp` (2100), `SetStrProp` (2109), `SetOrdProp` (2107), `SetFloatProp` (2105), `SetInt64Prop` (2105), `SetMethodProp` (2106)

78.7.56 StringToSet

Synopsis: Convert string description to a set.

Declaration: `function StringToSet(PropInfo: PPropInfo; const Value: string) : LongInt
function StringToSet(TypeInfo: PTypeInfo; const Value: string) : LongInt
procedure StringToSet(PropInfo: PPropInfo; const Value: string;
Result: Pointer)`


```
procedure StringToSet (TypeInfo: PTypeInfo; const Value: string;
                      Result: Pointer)
```

Visibility: default

Description: `StringToSet` converts the string representation of a set in `Value` to a integer representation of the set, using the property information found in `PropInfo`. This property information should point to the property information of a set property. The function returns the integer representation of the set. (i.e, the set value, typecast to an integer)

The string representation can be surrounded with square brackets, and must consist of the names of the elements of the base type of the set. The base type of the set should be an enumerated type. The elements should be separated by commas, and may be surrounded by spaces. each of the names will be fed to the `GetEnumValue` (2086) function.

For an example, see `SetToString` (2109).

Errors: No checking is done to see whether `PropInfo` points to valid property information. If a wrong name is given for an enumerated value, then an `EPropertyError` will be raised.

See also: `GetEnumName` (2084), `GetEnumValue` (2086), `SetToString` (2109)

78.8 TArrayTypeData

```
TArrayTypeData = packed record
private
    function GetElType : PTypeInfo
    ;
    function GetDims(aIndex: Byte) : PTypeInfo;
public
    property
        ElType : PTypeInfo;
        property Dims[Index: Byte]: PTypeInfo;
        Size
        : SizeInt;
        ElCount : SizeInt;
        ElTypeRef : PTypeInfo;
        DimCount
        : Byte;
        DimsRef : Array[0..255] of PTypeInfo;
end
```

`TArrayTypeData` is used to describe arrays in RTTI. It can be encountered when the type kind is `tkArray`, and is used for both static and dynamic arrays and single or multi-dimensional arrays. The type of the array elements is described in `elType`, and the ranges for each of the dimensions (specified in `DimCount` in `Dims`).

78.8.1 Property overview

Page	Properties	Access	Description
2113	<code>Dims</code>	<code>r</code>	Range information for each dimension in the array.
2113	<code>ElType</code>	<code>r</code>	Type information for an element in the array.

78.8.2 TArrayTypeData.ElType

Synopsis: Type information for an element in the array.

Declaration: `Property ElType : PTypeInfo`

Visibility: public

Access: Read

78.8.3 TArrayTypeData.Dims

Synopsis: Range information for each dimension in the array.

Declaration: `Property Dims[Index: Byte]: PTypeInfo`

Visibility: public

Access: Read

78.9 TClassData

```
TClassData = packed record
private
    function GetUnitName : ShortString
    ;
    function GetPropertyTable : PPropData;
public
    ClassType : TClass
    ;
    Parent : PTypeInfo;
    PropCount : SmallInt;
    property UnitName
    : ShortString;
    property PropertyTable : PPropData;
private
    UnitNameField
    : ShortString;
end
```

TClassData describes the memory layout of RTTI data generated for a class.

78.9.1 Property overview

Page	Properties	Access	Description
2114	PropertyTable	r	Property info for this class.
2113	UnitName	r	Name of unit in which class is implemente.

78.9.2 TClassData.UnitName

Synopsis: Name of unit in which class is implemente.

Declaration: `Property UnitName : ShortString`

Visibility: public

Access: Read

Description: `UnitName` returns the name of the unit in which the class is implemented.

78.9.3 TClassData.PropertyTable

Synopsis: Property info for this class.

Declaration: `Property PropertyTable : PPropData`

Visibility: public

Access: Read

Description: `PropertyTable` returns a pointer to a table with property RTTI.

78.10 TInterfaceData

```
TInterfaceData = packed record
private
    function GetUnitName : ShortString
    ;
    function GetPropertyTable : PPropData;
    function GetMethodTable
        : PIntfMethodTable;
public
    Parent : PTypeInfo;
    Flags : TIntfFlagsBase
    ;
    GUID : TGuid;
    property UnitName : ShortString;
    property PropertyTable
        : PPropData;
    property MethodTable : PIntfMethodTable;
private
    UnitNameField : ShortString;
end
```

`TInterfaceData` describes the memory layout of RTTI data generated for a COM interface. CORBA interfaces are described by `TInterfaceRawData` ([2116](#)).

78.10.1 Property overview

Page	Properties	Access	Description
2115	MethodTable	r	Method info for this interface.
2115	PropertyTable	r	Property info for this interface.
2115	UnitName	r	Unit name.

78.10.2 TInterfaceData.UnitName

Synopsis: Unit name.

Declaration: `Property UnitName : ShortString`

Visibility: public

Access: Read

Description: `UnitName` returns the unit name in which the interface is defined.

78.10.3 TInterfaceData.PropertyTable

Synopsis: Property info for this interface.

Declaration: `Property PropertyTable : PPropData`

Visibility: public

Access: Read

Description: `PropertyTable` returns a pointer to a table with property RTTI.

See also: `TInterfaceData.MethodTable` ([2115](#))

78.10.4 TInterfaceData.MethodTable

Synopsis: Method info for this interface.

Declaration: `Property MethodTable : PIntfMethodTable`

Visibility: public

Access: Read

Description: `MethodTable` returns a pointer to a table with method RTTI.

See also: `TInterfaceData.PropertyTable` ([2115](#))

78.11 TInterfaceRawData

```

TInterfaceRawData = packed record
private
    function GetUnitName
      : ShortString;
    function GetIIDStr : ShortString;
    function GetPropertyTable
      : PPropData;
    function GetMethodTable : PIntfMethodTable;
public
    Parent : PTypeInfo;
    Flags : TIntfFlagsBase;
    IID : TGuid;
    property UnitName : ShortString;
    property IIDStr : ShortString
    ;

```

```

    property PropertyTable : PPropData;
    property MethodTable : PIntfMethodTable
    ;
private
    UnitNameField : ShortString;
end

```

`TRawInterfaceData` describes the memory layout of RTTI data generated for a CORBA interface. COM interfaces are described by `TInterfaceData` ([2114](#)).

78.11.1 Property overview

Page	Properties	Access	Description
2116	<code>IIDStr</code>	r	GUID as string.
2117	<code>MethodTable</code>	r	Method info for this interface.
2116	<code>PropertyTable</code>	r	Property info for this interface.
2116	<code>UnitName</code>	r	Name of unit in which interface is defined.

78.11.2 TInterfaceRawData.UnitName

Synopsis: Name of unit in which interface is defined.

Declaration: `Property UnitName : ShortString`

Visibility: public

Access: Read

Description: `UnitName` returns the unit name in which the interface is defined.

78.11.3 TInterfaceRawData.IIDStr

Synopsis: GUID as string.

Declaration: `Property IIDStr : ShortString`

Visibility: public

Access: Read

Description: `IIDStr` returns a string representation of IID.

78.11.4 TInterfaceRawData.PropertyTable

Synopsis: Property info for this interface.

Declaration: `Property PropertyTable : PPropData`

Visibility: public

Access: Read

Description: `PropertyTable` returns a pointer to a table with property RTTI.

See also: `TInterfaceRawData.MethodTable` ([2117](#))

78.11.5 TInterfaceRawData.MethodTable

Synopsis: Method info for this interface.

Declaration: `Property MethodTable : PIntfMethodTable`

Visibility: `public`

Access: `Read`

Description: `MethodTable` returns a pointer to a table with method RTTI.

See also: `TInterfaceRawData.PropertyTable` ([2116](#))

78.12 TIntfMethodEntry

```

TIntfMethodEntry = packed record
private
    function GetParam(Index
        : Word) : PVmtMethodParam;
    function GetResultLocs : PParameterLocations
    ;
    function GetTail : Pointer;
    function GetNext : TIntfMethodEntry
    ;
    function GetName : ShortString;
public
    ResultType : PTypeInfo
    ;
    CC : TCallConv;
    Kind : TMethodKind;
    ParamCount : Word;
    StackSize
        : SizeInt;
    NamePtr : PShortString;
    property Name : ShortString
    ;
    property Param[Index: Word]: PVmtMethodParam;
    property ResultLocs
        : PParameterLocations;
    property Tail : Pointer;
    property Next
        : TIntfMethodEntry;
end

```

`TIntfMethodEntry` is a description of the RTTI generated for an `Interface` method. This is different from class or record method RTTI info, which is described by `TVmtMethodEntry` ([2081](#))

78.12.1 Property overview

Page	Properties	Access	Description
2118	Name	r	Name of the method.
2119	Next	r	Pointer next <code>TIntfMethodEntry</code> entry in array.
2118	Param	r	Indexed access to parameter descriptions.
2118	ResultLocs	r	Result location descriptions.
2119	Tail	r	Pointer to location after <code>TIntfMethodEntry</code> data.

78.12.2 TIntfMethodEntry.Name

Synopsis: Name of the method.

Declaration: `Property Name : ShortString`

Visibility: public

Access: Read

Description: `Name` dereferences the `NamePtr` field and returns the name of the method.

See also: `TIntfMethodEntry.Param` ([2118](#))

78.12.3 TIntfMethodEntry.Param

Synopsis: Indexed access to parameter descriptions.

Declaration: `Property Param[Index: Word]: PVmtMethodParam`

Visibility: public

Access: Read

Description: `Param` offers indexed access to the parameter descriptions. It returns the location of the `Index`-th parameter of the method. `Index` must be in the range 0 to `ParamCount-1` (included).

See also: `TIntfMethodEntry.ResultLocs` ([2118](#))

78.12.4 TIntfMethodEntry.ResultLocs

Synopsis: Result location descriptions.

Declaration: `Property ResultLocs : PParameterLocations`

Visibility: public

Access: Read

Description: `ResultLocs` points to a record with result location descriptors: this describes where the result(s) will be written.

See also: `TParameterLocations` ([2122](#))

78.12.5 TIntfMethodEntry.Tail

Synopsis: Pointer to location after TIntfMethodEntry data.

Declaration: `Property Tail : Pointer`

Visibility: `public`

Access: `Read`

Description: `Tail` returns a pointer to the memory location after the `TIntfMethodEntry` data, which can be again a `TIntfMethodEntry` record or other RTTI Data.

See also: `TIntfMethodEntry.Next` ([2119](#))

78.12.6 TIntfMethodEntry.Next

Synopsis: Pointer next TIntfMethodEntry entry in array.

Declaration: `Property Next : PIntfMethodEntry`

Visibility: `public`

Access: `Read`

Description: `Next` is the same as `TIntfMethodEntry.Tail` ([2119](#)), but is typed: it returns a pointer to the next `TIntfMethodEntry` record in the RTTI.

See also: `TIntfMethodEntry.Tail` ([2119](#))

78.13 TIntfMethodTable

```
TIntfMethodTable = packed record
private
  function GetMethod(Index
    : Word) : PIntfMethodEntry;
public
  Count : Word;
  RTTICount : Word
  ;
  property Method[Index: Word]: PIntfMethodEntry;
end
```

`TIntfMethodTable` is a record structure describing all the methods of an interface for which RTTI is generated. It is basically an array of pointers `TIntfMethodTable.Method` ([2120](#)).

78.13.1 Property overview

Page	Properties	Access	Description
2120	Method	r	indexed access to the methods of the interface.

78.13.2 TIntfMethodTable.Method

Synopsis: indexed access to the methods of the interface.

Declaration: `Property Method[Index: Word]: PIntfMethodEntry`

Visibility: public

Access: Read

Description: `Method` offers indexed access to the method descriptions. It returns the location of the `Index`-th method of the interface. `Index` must be in the range 0 to `Count-1` (included).

See also: `PIntfMethodEntry` ([2074](#)), `TIntfMethodEntry` ([2117](#))

78.14 TManagedField

```
TManagedField = packed record
private
    function GetTypeRef : PTypeInfo
    ;
public
    property TypeRef : PTypeInfo;
    TypeRefRef : PTypeInfo
    ;
    FldOffset : SizeInt;
end
```

`TManagedField` describes 1 managed field in a record. It consists of type information (`TypeRef`) and an offset in the record's memory layout (`FldOffset`). Size can be determined from the type information.

78.14.1 Property overview

Page	Properties	Access	Description
2120	<code>TypeRef</code>	r	Type information for the field.

78.14.2 TManagedField.TypeRef

Synopsis: Type information for the field.

Declaration: `Property TypeRef : PTypeInfo`

Visibility: public

Access: Read

78.15 TParameterLocation

```
TParameterLocation = packed record
private
    LocType : Byte;
    function
```

```

    GetRegType : TRegisterType;
    function GetReference : Boolean;
    function GetShiftVal : Int8;
public
    RegSub : TSubRegister;
    RegNumber
    : Word;
    Offset : SizeInt;
    property Reference : Boolean;
    property
    RegType : TRegisterType;
    property ShiftVal : Int8;
end

```

`TParameterLocation` is used by RTTI to describe the location of a parameter (argument) when arguments are passed to a method call. It has the same memory layout as the actual RTTI data. It offers the following fields and properties:

RegSub Sub register indicator.

RegNumber Register number.

Offset Stack offset or index of register.

Property Reference Is the offset a stack offset or register index.

Property RegType Register type.

Property ShiftVal Register shift.

78.15.1 Property overview

Page	Properties	Access	Description
2121	Reference	r	Is the offset a stack offset or register index.
2122	RegType	r	Register type.
2122	ShiftVal	r	Register shift.

78.15.2 TParameterLocation.Reference

Synopsis: Is the offset a stack offset or register index.

Declaration: `Property Reference : Boolean`

Visibility: public

Access: Read

Description: `Reference` indicates whether the register is the index register (`True`), otherwise the register in which (part of) the parameter resides.

See also: `TParameterLocation.ShiftVal` ([2122](#))

78.15.3 TParameterLocation.RegType

Synopsis: Register type.

Declaration: `Property RegType : TRegisterType`

Visibility: public

Access: Read

Description: `RegType` returns the type of the register location. It is calculated from a private field.

See also: `TRegisterType` ([2079](#))

78.15.4 TParameterLocation.ShiftVal

Synopsis: Register shift.

Declaration: `Property ShiftVal : Int8`

Visibility: public

Access: Read

Description: `ShiftVal` is zero if `Reference` ([2121](#)) is false, or `Offset` otherwise.

See also: `Reference` ([2121](#))

78.16 TParameterLocations

```
TParameterLocations = packed record
private
  function GetLocation
    (aIndex: Byte) : PParameterLocation;
  function GetTail : Pointer
  ;
public
  Count : Byte;
  property Location[Index: Byte]: PParameterLocation
  ;
  property Tail : Pointer;
end
```

`TParameterLocations` is used by RTTI to describe the array of locations of method parameters (arguments) when arguments are passed to a method call. It describes the memory layout of the actual RTTI data. It offers the following fields and properties:

78.16.1 Property overview

Page	Properties	Access	Description
2123	Location	r	Indexed access to the various parameter location descriptors.
2123	Tail	r	Memory location after parameter data.

78.16.2 TParameterLocations.Location

Synopsis: Indexed access to the various parameter location descriptors.

Declaration: `Property Location[Index: Byte]: PParameterLocation`

Visibility: public

Access: Read

Description: `Location` offers indexed access to the various `TParameterLocation` (2121) records that make up a method's parameters description. Note that it returns a pointer to the actual record, not a copy of the record.

See also: `PParameterLocations` (2074), `TParameterLocation` (2121), `PParameterLocation` (2074)

78.16.3 TParameterLocations.Tail

Synopsis: Memory location after parameter data.

Declaration: `Property Tail : Pointer`

Visibility: public

Access: Read

Description: `Tail` points to the memory location after the `TParameterLocations` data (the last `TParameterLocation` (2121) record). It can be used to get the location of the next structure in the RTTI.

See also: `TParameterLocation` (2121)

78.17 TProcedureParam

```
TProcedureParam = packed record
private
  function GetParamType :
    PTypeInfo;
  function GetFlags : Byte;
public
  property ParamType
    : PTypeInfo;
  property Flags : Byte;
  ParamFlags : TParamFlags
  ;
  ParamTypeRef : PTypeInfo;
  Name : ShortString;
end
```

`TProcedureParam` describes a single parameter to a procedure (or function).

78.17.1 Property overview

Page	Properties	Access	Description
2124	Flags	r	Flags for this parameter (see <code>TParamFlags</code>).
2124	ParamType	r	Type information for this parameter.

78.17.2 TProcedureParam.ParamType

Synopsis: Type information for this parameter.

Declaration: `Property ParamType : PTypeInfo`

Visibility: `public`

Access: `Read`

78.17.3 TProcedureParam.Flags

Synopsis: Flags for this parameter (see `TParamFlags`).

Declaration: `Property Flags : Byte`

Visibility: `public`

Access: `Read`

78.18 TProcedureSignature

```
TProcedureSignature = packed record
private
    function GetResultType
        : PTypeInfo;
public
    property ResultType : PTypeInfo;
    Flags :
        Byte;
    CC : TCallConv;
    ResultTypeRef : PTypeInfo;
    ParamCount
        : Byte;
    function GetParam(ParamIndex: Integer) : PProcedureParam
        ;
end
```

`TProcedureSignature` describes a procedure/method call signature. It consists of some flags (`Flags`), a calling convention (`CC`), the result type (`ResultType`) if any, and a list of `ParamCount` parameters (of type `TProcedureParam` ([2123](#))).

78.18.1 Method overview

Page	Method	Description
2125	<code>GetParam</code>	Get parameter signature.

78.18.2 Property overview

Page	Properties	Access	Description
2125	<code>ResultType</code>	<code>r</code>	Result type info (Nil if no result).

78.18.3 TProcedureSignature.GetParam

Synopsis: Get parameter signature.

Declaration: `function GetParam(ParamIndex: Integer) : PProcedureParam`

Visibility: public

Description: `GetParam` can be used to retrieve a pointer to the description of a parameter. The index `ParamIndex` is zero-based.

Errors: In case of an invalid parameter index, `Nil` is returned.

See also: `TProcedureParam` ([2123](#))

78.18.4 TProcedureSignature.ResultType

Synopsis: Result type info (`Nil` if no result).

Declaration: `Property ResultType : PTypeInfo`

Visibility: public

Access: Read

78.19 TPropData

```
TPropData = packed record
private
    function GetProp(Index: Word)
        : PPropInfo;
    function GetTail : Pointer;
public
    PropCount : Word
    ;
    PropList : record
    public
        _alignmentdummy : PtrInt;
    end
    ;
    property Prop[Index: Word]: PPropInfo;
    property Tail : Pointer
    ;
end
```

The `TPropData` record is not used, but is provided for completeness and compatibility with Delphi.

78.19.1 Property overview

Page	Properties	Access	Description
2126	Prop	r	Indexed access to property type info.
2126	Tail	r	Pointer to next RTTI structure.

78.19.2 TPropData.Prop

Synopsis: Indexed access to property type info.

Declaration: `Property Prop[Index: Word]: PPropInfo`

Visibility: public

Access: Read

Description: `Prop` provides indexed access to the property type info of all properties of a class. Valid values for `Index` are in the range `0..WordCount-1`

See also: `PPropInfo` ([2074](#)), `TPropInfo` ([2127](#))

78.19.3 TPropData.Tail

Synopsis: Pointer to next RTTI structure.

Declaration: `Property Tail : Pointer`

Visibility: public

Access: Read

Description: `Tail` points to the memory area after the last property info. It can be used to get the next RTTI info.

78.20 TPropInfo

```

TPropInfo = packed record
private
    function GetPropType : PTypeInfo
    ;
    function GetTail : Pointer;
    function GetNext : PPropInfo;
public
    PropTypeRef : PTypeInfo;
    GetProc : CodePointer;
    SetProc :
    CodePointer;
    StoredProc : CodePointer;
    Index : Integer;
    Default
    : LongInt;
    NameIndex : SmallInt;
    PropProcs : Byte;
    Name : ShortString
    ;
    property PropType : PTypeInfo;
    property Tail : Pointer;
    property
    Next : PPropInfo;
end

```

The `TPropInfo` record describes one published property of a class. The property information of a class are stored as an array of `TPropInfo` records.

The `Name` field is stored not with 255 characters, but with just as many characters as required to store the name.

78.20.1 Property overview

Page	Properties	Access	Description
2127	<code>Next</code>	<code>r</code>	Typed pointer <code>TPropInfo</code> record after current <code>TPropInfo</code> data.
2127	<code>PropType</code>	<code>r</code>	Property type.
2127	<code>Tail</code>	<code>r</code>	Pointer to memory location after <code>TPropInfo</code> data.

78.20.2 TPropInfo.PropertyType

Synopsis: Property type.

Declaration: `Property PropType : PTypeInfo`

Visibility: `public`

Access: `Read`

78.20.3 TPropInfo.Tail

Synopsis: Pointer to memory location after `TPropInfo` data.

Declaration: `Property Tail : Pointer`

Visibility: `public`

Access: `Read`

Description: `Tail` points to the memory location right after the `TPropInfo` data. This can be another `TPropInfo` record, or a completely different kind of information.

See also: `TPropInfo.Next` ([2127](#))

78.20.4 TPropInfo.Next

Synopsis: Typed pointer `TPropInfo` record after current `TPropInfo` data.

Declaration: `Property Next : PPropInfo`

Visibility: `public`

Access: `Read`

Description: `Next` points to the same memory location as `TPropInfo.Tail` ([2127](#)), but is typed. It should not be used on the last element of a `TPropInfo` array.

See also: `TPropInfo.Tail` ([2127](#))

78.21 TTypeData

```

TTypeData = packed record
private
    function GetBaseType : PTypeInfo
    ;
    function GetCompType : PTypeInfo;
    function GetParentInfo : PTypeInfo
    ;
    function GetRecInitData : PRecInitData;
    function GetHelperParent
    : PTypeInfo;
    function GetExtendedInfo : PTypeInfo;
    function GetIntfParent
    : PTypeInfo;
    function GetRawIntfParent : PTypeInfo;
    function
    GetIIDStr : ShortString;
    function GetElType : PTypeInfo;
    function
    GetElType2 : PTypeInfo;
    function GetInstanceType : PTypeInfo;
    function GetRefType : PTypeInfo;
public
    property BaseType : PTypeInfo
    ;
    property CompType : PTypeInfo;
    property ParentInfo : PTypeInfo
    ;
    property RecInitData : PRecInitData;
    property HelperParent
    : PTypeInfo;
    property ExtendedInfo : PTypeInfo;
    property IntfParent
    : PTypeInfo;
    property RawIntfParent : PTypeInfo;
    property IIDStr
    : ShortString;
    property ElType2 : PTypeInfo;
    property ElType
    : PTypeInfo;
    property InstanceType : PTypeInfo;
    property RefType
    : PTypeInfo;
case TTypeKind of
tkUnknown: (
);, tkLString: (
);,
    tkWString: (
);, tkVariant: (
);, tkUString: (
);
tkAString: (

```

```

public
  CodePage : Word;
);
tkInt64: (
public
  OrdType : TOrdType;
case
  TTypeKind of
tkInteger: (
public
  MinValue : LongInt;
  MaxValue
    : LongInt;
case TTypeKind of
tkEnumeration: (
public
  BaseTypeRef
    : PPTypInfo;
  NameList : ShortString;
);
);, tkChar: (
public
  MinValue : LongInt;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration
  : (
public
  BaseTypeRef : PPTypInfo;
  NameList : ShortString;
);
);, tkEnumeration: (
public
  MinValue : LongInt;
  MaxValue :
    LongInt;
case TTypeKind of
tkEnumeration: (
public
  BaseTypeRef
    : PPTypInfo;
  NameList : ShortString;
);
);, tkBool: (
public
  MinValue : LongInt;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration
  : (
public
  BaseTypeRef : PPTypInfo;
  NameList : ShortString;
);
);

```

```

);, tkWChar: (
public
    MinValue : LongInt;
    MaxValue : LongInt
    ;
case TTypeKind of
tkEnumeration: (
public
    BaseTypeRef : PTypeInfo
    ;
    NameList : ShortString;
);
);
tkInt64: (
public
    MinInt64Value
        : Int64;
    MaxInt64Value : Int64;
);
tkQWord: (
public
    MinQWordValue
        : QWord;
    MaxQWordValue : QWord;
);
tkSet: (
public
    SetSize :
        SizeInt;
    CompTypeRef : PTypeInfo;
);
);, tkQWord: (
public
    OrdType
        : TOrdType;
case TTypeKind of
tkInteger: (
public
    MinValue : LongInt
    ;
    MaxValue : LongInt;
case TTypeKind of
tkEnumeration: (
public
    BaseTypeRef : PTypeInfo;
    NameList : ShortString;
);
);, tkChar
    : (
public
    MinValue : LongInt;
    MaxValue : LongInt;
case TTypeKind
    of

```

```

tkEnumeration: (
public
  BaseTypeRef : PTypeInfo;
  NameList
    : ShortString;
);
);, tkEnumeration: (
public
  MinValue : LongInt
    ;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration: (
public
  BaseTypeRef : PTypeInfo;
  NameList : ShortString;
);
);, tkBool
  : (
public
  MinValue : LongInt;
  MaxValue : LongInt;
case TTypeKind
  of
tkEnumeration: (
public
  BaseTypeRef : PTypeInfo;
  NameList
    : ShortString;
);
);, tkWChar: (
public
  MinValue : LongInt;
  MaxValue
    : LongInt;
case TTypeKind of
tkEnumeration: (
public
  BaseTypeRef
    : PTypeInfo;
  NameList : ShortString;
);
);
tkInt64: (
public
  MinInt64Value : Int64;
  MaxInt64Value : Int64;
);
tkQWord: (
public
  MinQWordValue : QWord;
  MaxQWordValue : QWord;
);
tkSet: (

```

```

public
  SetSize : SizeInt;
  CompTypeRef : PPTypeInfo;
);
);, tkInteger
  : (
public
  OrdType : TOrdType;
case TTypeKind of
tkInteger: (
public
  MinValue : LongInt;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration
  : (
public
  BaseTypeRef : PPTypeInfo;
  NameList : ShortString;
);
);, tkChar: (
public
  MinValue : LongInt;
  MaxValue : LongInt
  ;
case TTypeKind of
tkEnumeration: (
public
  BaseTypeRef : PPTypeInfo
  ;
  NameList : ShortString;
);
);, tkEnumeration: (
public
  MinValue
  : LongInt;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration
  : (
public
  BaseTypeRef : PPTypeInfo;
  NameList : ShortString;
);
);, tkBool: (
public
  MinValue : LongInt;
  MaxValue : LongInt
  ;
case TTypeKind of
tkEnumeration: (
public
  BaseTypeRef : PPTypeInfo
  ;

```

```

    NameList : ShortString;
);
);, tkWChar: (
public
    MinValue :
        LongInt;
    MaxValue : LongInt;
case TTypeKind of
tkEnumeration:
    (
public
    BaseTypeRef : PTypeInfo;
    NameList : ShortString;
);
);
tkInt64: (
public
    MinInt64Value : Int64;
    MaxInt64Value : Int64
    ;
);
tkQWord: (
public
    MinQWordValue : QWord;
    MaxQWordValue :
        QWord;
);
tkSet: (
public
    SetSize : SizeInt;
    CompTypeRef : PTypeInfo
    ;
);
);, tkChar: (
public
    OrdType : TOrdType;
case TTypeKind of
    tkInteger: (
public
        MinValue : LongInt;
        MaxValue : LongInt;
case
    TTypeKind of
tkEnumeration: (
public
    BaseTypeRef : PTypeInfo;
    NameList : ShortString;
);
);, tkChar: (
public
    MinValue : LongInt
    ;
    MaxValue : LongInt;
case TTypeKind of

```

```

tkEnumeration: (
public
  BaseTypeRef : PTypeInfo;
  NameList : ShortString;
);
);, tkEnumeration
  : (
public
  MinValue : LongInt;
  MaxValue : LongInt;
case TTypeKind
of
tkEnumeration: (
public
  BaseTypeRef : PTypeInfo;
  NameList
  : ShortString;
);
);, tkBool: (
public
  MinValue : LongInt;
  MaxValue
  : LongInt;
case TTypeKind of
tkEnumeration: (
public
  BaseTypeRef
  : PTypeInfo;
  NameList : ShortString;
);
);, tkWChar: (
public
  MinValue : LongInt;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration
  : (
public
  BaseTypeRef : PTypeInfo;
  NameList : ShortString;
  );
);
tkInt64: (
public
  MinInt64Value : Int64;
  MaxInt64Value
  : Int64;
);
tkQWord: (
public
  MinQWordValue : QWord;
  MaxQWordValue
  : QWord;
);

```

```

tkSet: (
public
  SetSize : SizeInt;
  CompTypeRef :
  PPTypeInfo;
);
);, tkEnumeration: (
public
  OrdType : TOrdType;
  case TTypeKind of
tkInteger: (
public
  MinValue : LongInt;
  MaxValue
  : LongInt;
case TTypeKind of
tkEnumeration: (
public
  BaseTypeRef
  : PPTypeInfo;
  NameList : ShortString;
);
);, tkChar: (
public
  MinValue : LongInt;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration
  : (
public
  BaseTypeRef : PPTypeInfo;
  NameList : ShortString;
  );
);, tkEnumeration: (
public
  MinValue : LongInt;
  MaxValue :
  LongInt;
case TTypeKind of
tkEnumeration: (
public
  BaseTypeRef
  : PPTypeInfo;
  NameList : ShortString;
);
);, tkBool: (
public
  MinValue : LongInt;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration
  : (
public
  BaseTypeRef : PPTypeInfo;

```



```

    NameList : ShortString;
  );
);, tkWChar: (
public
  MinValue : LongInt;
  MaxValue : LongInt
  ;
case TTypeKind of
tkEnumeration: (
public
  BaseTypeRef : PTypeInfo
  ;
  NameList : ShortString;
);
);
tkInt64: (
public
  MinInt64Value
    : Int64;
  MaxInt64Value : Int64;
);
tkQWord: (
public
  MinQWordValue
    : QWord;
  MaxQWordValue : QWord;
);
tkSet: (
public
  SetSize :
  SizeInt;
  CompTypeRef : PTypeInfo;
);
);, tkBool: (
public
  OrdType
    : TOrdType;
case TTypeKind of
tkInteger: (
public
  MinValue : LongInt
  ;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration: (
public
  BaseTypeRef : PTypeInfo;
  NameList : ShortString;
);
);, tkChar
  : (
public
  MinValue : LongInt;
  MaxValue : LongInt;

```

```

case TTypeKind
  of
tkEnumeration: (
public
  BaseTypeRef : PTypeInfo;
  NameList
    : ShortString;
);
);, tkEnumeration: (
public
  MinValue : LongInt
    ;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration: (
public
  BaseTypeRef : PTypeInfo;
  NameList : ShortString;
);
);, tkBool
  : (
public
  MinValue : LongInt;
  MaxValue : LongInt;
case TTypeKind
  of
tkEnumeration: (
public
  BaseTypeRef : PTypeInfo;
  NameList
    : ShortString;
);
);, tkWChar: (
public
  MinValue : LongInt;
  MaxValue
    : LongInt;
case TTypeKind of
tkEnumeration: (
public
  BaseTypeRef
    : PTypeInfo;
  NameList : ShortString;
);
);
tkInt64: (
public
  MinInt64Value : Int64;
  MaxInt64Value : Int64;
);
tkQWord: (
public
  MinQWordValue : QWord;
  MaxQWordValue : QWord;

```

```

);
tkSet: (
public
    SetSize : SizeInt;
    CompTypeRef : PPTypInfo;
);
);, tkWChar:
(
public
    OrdType : TOrdType;
case TTypeKind of
tkInteger: (
public
    MinValue : LongInt;
    MaxValue : LongInt;
case TTypeKind of
tkEnumeration
    : (
public
    BaseTypeRef : PPTypInfo;
    NameList : ShortString;
    );
);, tkChar: (
public
    MinValue : LongInt;
    MaxValue : LongInt
    ;
case TTypeKind of
tkEnumeration: (
public
    BaseTypeRef : PPTypInfo
    ;
    NameList : ShortString;
);
);, tkEnumeration: (
public
    MinValue
    : LongInt;
    MaxValue : LongInt;
case TTypeKind of
tkEnumeration
    : (
public
    BaseTypeRef : PPTypInfo;
    NameList : ShortString;
    );
);, tkBool: (
public
    MinValue : LongInt;
    MaxValue : LongInt
    ;
case TTypeKind of
tkEnumeration: (
public

```

```

    BaseTypeRef : PTypeInfo
    ;
    NameList : ShortString;
);
);, tkWChar: (
public
    MinValue :
    LongInt;
    MaxValue : LongInt;
case TTypeKind of
tkEnumeration:
    (
public
    BaseTypeRef : PTypeInfo;
    NameList : ShortString;
);
);
tkInt64: (
public
    MinInt64Value : Int64;
    MaxInt64Value : Int64
    ;
);
tkQWord: (
public
    MinQWordValue : QWord;
    MaxQWordValue :
    QWord;
);
tkSet: (
public
    SetSize : SizeInt;
    CompTypeRef : PTypeInfo
    ;
);
);, tkSet: (
public
    OrdType : TOrdType;
case TTypeKind of
tkInteger
    : (
public
    MinValue : LongInt;
    MaxValue : LongInt;
case TTypeKind
    of
tkEnumeration: (
public
    BaseTypeRef : PTypeInfo;
    NameList
    : ShortString;
);
);, tkChar: (
public

```

```

    MinValue : LongInt;
    MaxValue
      : LongInt;
case TTypeKind of
tkEnumeration: (
public
  BaseTypeRef
    : PPTypeInfo;
  NameList : ShortString;
);
);, tkEnumeration: (
public
  MinValue : LongInt;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration
  : (
public
  BaseTypeRef : PPTypeInfo;
  NameList : ShortString;
  );
);, tkBool: (
public
  MinValue : LongInt;
  MaxValue : LongInt
  ;
case TTypeKind of
tkEnumeration: (
public
  BaseTypeRef : PPTypeInfo
  ;
  NameList : ShortString;
);
);, tkWChar: (
public
  MinValue :
  LongInt;
  MaxValue : LongInt;
case TTypeKind of
tkEnumeration:
  (
public
  BaseTypeRef : PPTypeInfo;
  NameList : ShortString;
);
  );
tkInt64: (
public
  MinInt64Value : Int64;
  MaxInt64Value : Int64
  ;
);
tkQWord: (
public

```

```

    MinQWordValue : QWord;
    MaxQWordValue :
    QWord;
);
tkSet: (
public
    SetSize : SizeInt;
    CompTypeRef : PPTypeInfo
    ;
);
);
tkFloat: (
public
    FloatType : TFloatType;
);
tkSSString:
(
public
    MaxLength : Byte;
);
tkClass: (
public
    ClassType : TClass
    ;
    ParentInfoRef : PPTypeInfo;
    PropCount : SmallInt;
    UnitName
    : ShortString;
);
tkRecord: (
public
    RecInitInfo : Pointer;
    RecSize
    : Integer;
case Boolean of
False: (
public
    ManagedFldCount : Integerdeprecated
    ;
);
True: (
public
    TotalFieldCount : Integer;
);
);
tkHelper:
(
public
    HelperParentRef : PPTypeInfo;
    ExtendedInfoRef : PPTypeInfo
    ;
    HelperProps : SmallInt;
    HelperUnit : ShortString;
);

```

```

tkMethod
  : (
public
  MethodKind : TMethodKind;
  ParamCount : Byte;
  ParamList
    : Array[0..1023] of Char;
);
tkProcVar: (
public
  ProcSig : TProcedureSignature
  ;
);
tkInterface: (
public
  IntfParentRef : PTypeInfo;
  IntfFlags
    : TIntfFlagsBase;
  GUID : TGuid;
  IntfUnit : ShortString;
);
tkInterfaceRaw
  : (
public
  RawIntfParentRef : PTypeInfo;
  RawIntfFlags : TIntfFlagsBase
  ;
  IID : TGuid;
  RawIntfUnit : ShortString;
);
tkArray: (
public
  ArrayData : TArrayTypeData;
);
tkDynArray: (
public
  elSize :
  PtrUInt;
  elType2Ref : PTypeInfo;
  varType : LongInt;
  elTypeRef
    : PTypeInfo;
  DynUnitName : ShortStringBase;
);
tkClassRef: (
public
  InstanceTypeRef : PTypeInfo;
);
tkPointer: (
public
  RefTypeRef
    : PTypeInfo;
);
end

```

If the `TypeInfo` kind is `tkClass`, then the property information follows the `UnitName` string, as an array of `TPropInfo` (2127) records.

78.21.1 Property overview

Page	Properties	Access	Description
2143	<code>BaseType</code>	r	Base type on which this type is based.
2143	<code>CompType</code>	r	Comp type.
2145	<code>ElType</code>	r	Dynamical array Element type.
2145	<code>ElType2</code>	r	Dynamical array Element type.
2144	<code>ExtendedInfo</code>	r	Type information for the extended type (record or class).
2144	<code>HelperParent</code>	r	Type information for parent helper class.
2145	<code>IIDStr</code>	r	IID string representation of interface.
2145	<code>InstanceType</code>	r	Pointer to instance type data.
2144	<code>IntfParent</code>	r	Parent interface type data.
2143	<code>ParentInfo</code>	r	Parent class type info.
2145	<code>RawIntfParent</code>	r	Raw interface parent.
2144	<code>RecInitData</code>	r	Pointer to record initialization data.
2145	<code>RefType</code>	r	Type info for referred type (Nil for untyped pointer).

78.21.2 TTypeData.BaseType

Synopsis: Base type on which this type is based.

Declaration: `Property BaseType : PTypeInfo`

Visibility: public

Access: Read

78.21.3 TTypeData.CompType

Synopsis: Comp type.

Declaration: `Property CompType : PTypeInfo`

Visibility: public

Access: Read

78.21.4 TTypeData.ParentInfo

Synopsis: Parent class type info.

Declaration: `Property ParentInfo : PTypeInfo`

Visibility: public

Access: Read

78.21.5 TTypeData.RecInitData

Synopsis: Pointer to record initialization data.

Declaration: `Property RecInitData : PRecInitData`

Visibility: public

Access: Read

Description: `RecInitData` is a pointer to a `TRecInitData` (2079) record. It is only valid if the type data describes a record.

See also: `TRecInitData` (2079)

78.21.6 TTypeData.HelperParent

Synopsis: Type information for parent helper class.

Declaration: `Property HelperParent : PTypeInfo`

Visibility: public

Access: Read

Description: `HelperParent` points to the type information of the parent helper class. It is `Nil` if there is no parent class.

See also: `TTypeData.ExtendedInfo` (2144)

78.21.7 TTypeData.ExtendedInfo

Synopsis: Type information for the extended type (record or class).

Declaration: `Property ExtendedInfo : PTypeInfo`

Visibility: public

Access: Read

Description: `ExtendedInfo` points to the type information of the type that is being extended.

See also: `TTypeData.HelperParent` (2144)

78.21.8 TTypeData.IntfParent

Synopsis: Parent interface type data.

Declaration: `Property IntfParent : PTypeInfo`

Visibility: public

Access: Read

78.21.9 TTypeData.RawIntfParent

Synopsis: Raw interface parent.

Declaration: `Property RawIntfParent : PTypeInfo`

Visibility: public

Access: Read

78.21.10 TTypeData.IIDStr

Synopsis: IID string representation of interface.

Declaration: `Property IIDStr : ShortString`

Visibility: public

Access: Read

78.21.11 TTypeData.ElType2

Synopsis: Dynamical array Element type.

Declaration: `Property ElType2 : PTypeInfo`

Visibility: public

Access: Read

78.21.12 TTypeData.ElType

Synopsis: Dynamical array Element type.

Declaration: `Property ElType : PTypeInfo`

Visibility: public

Access: Read

78.21.13 TTypeData.InstanceType

Synopsis: Pointer to instance type data.

Declaration: `Property InstanceType : PTypeInfo`

Visibility: public

Access: Read

78.21.14 TTypeData.RefType

Synopsis: Type info for referred type (Nil for untyped pointer).

Declaration: `Property RefType : PTypeInfo`

Visibility: public

Access: Read

78.22 TVmtFieldEntry

```
TVmtFieldEntry = packed record
private
    function GetNext : PVmtFieldEntry
    ;
    function GetTail : Pointer;
public
    FieldOffset : PtrUInt;
    TypeIndex : Word;
    Name : ShortString;
    property Tail : Pointer
    ;
    property Next : PVmtFieldEntry;
end
```

TVmtFieldEntry records are generated by the compiler for all fields of a record or class that have RTTI associated with them. They describe the field as known to the compiler.

78.22.1 Property overview

Page	Properties	Access	Description
2146	Next	r	Pointer to the next TVmtFieldEntry.
2146	Tail	r	Pointer to the next RTTI structure.

78.22.2 TVmtFieldEntry.Tail

Synopsis: Pointer to the next RTTI structure.

Declaration: `Property Tail : Pointer`

Visibility: public

Access: Read

Description: Tail points to the memory location after the TVmtFieldEntry record. This can be another TVmtFieldEntry record or an entirely different data structure.

See also: TVmtFieldEntry.Next ([2146](#))

78.22.3 TVmtFieldEntry.Next

Synopsis: Pointer to the next TVmtFieldEntry.

Declaration: `Property Next : PVmtFieldEntry`

Visibility: public

Access: Read

Description: Next is the same as TVmtFieldEntry.Tail ([2146](#)), but is typed. Note that on the last TVmtFieldEntry in the array, the next element is not a TVmtFieldEntry.

See also: TVmtFieldEntry.Tail ([2146](#))

78.23 TVmtFieldTable

```

TVmtFieldTable = packed record
private
    function GetField(aIndex
        : Word) : PVmtFieldEntry;
public
    Count : Word;
    ClassTab : PVmtFieldClassTab
    ;
    Fields : Array[0..0] of TVmtFieldEntry;
    property Field[aIndex
        : Word]: PVmtFieldEntry;
end

```

TVmtFieldTable describes the fields for which RTTI was generated. A TVmtFieldTable entry is generated by the compiler in the RTTI information, it is not something one creates manually. Basically it contains a list of TVmtFieldEntry (2146) values.

78.23.1 Property overview

Page	Properties	Access	Description
2147	Field	r	Indexed access to the fields in the table.

78.23.2 TVmtFieldTable.Field

Synopsis: Indexed access to the fields in the table.

Declaration: Property Field[aIndex: Word]: PVmtFieldEntry

Visibility: public

Access: Read

Description: Field returns the location of the aIndexthe field in the table.

See also: TVmtFieldEntry ([2146](#))

78.24 TVmtMethodParam

```

TVmtMethodParam = packed record
private
    function GetTail : Pointer
    ;
    function GetNext : PVmtMethodParam;
    function GetName : ShortString
    ;
public
    ParamType : PTypeInfo;
    Flags : TParamFlags;
    NamePtr
        : PShortString;
    ParaLocs : PParameterLocations;

```

```

property Name
: ShortString;
property Tail : Pointer;
property Next : PVmtMethodParam
;
end

```

TVmtMethodParam describes the type and location of a parameter of a method.

78.24.1 Property overview

Page	Properties	Access	Description
2148	Name	r	Parameter name.
2148	Next	r	Pointer to the next TVmtMethodParam record.
2148	Tail	r	Pointer to location after TVmtMethodParam data.

78.24.2 TVmtMethodParam.Name

Synopsis: Parameter name.

Declaration: `Property Name : ShortString`

Visibility: public

Access: Read

Description: Name is an ease-of-access property for the NamePtr field, it simply returns the name as a string.

78.24.3 TVmtMethodParam.Tail

Synopsis: Pointer to location after TVmtMethodParam data.

Declaration: `Property Tail : Pointer`

Visibility: public

Access: Read

Description: Tail points to the memory location just after the TVmtMethodParam structure. It can be used to get access to the next RTTI data, which can be again a TVmtMethodParam structure or another structure.

See also: TVmtMethodParam.Next ([2148](#))

78.24.4 TVmtMethodParam.Next

Synopsis: Pointer to the next TVmtMethodParam record.

Declaration: `Property Next : PVmtMethodParam`

Visibility: public

Access: Read

Description: Next is the same as TVmtMethodParam.Tail ([2148](#)), but is typed: it returns a pointer to the next TVmtMethodParam record in the RTTI.

See also: TVmtMethodParam.Tail ([2148](#))

78.25 TVmtMethodTable

```

TVmtMethodTable = packed record
private
    function GetEntry(Index
        : LongWord) : PVmtMethodEntry;
public
    Count : LongWord;
    property
        Entry[Index: LongWord]: PVmtMethodEntry;
private
    Entries : Array
        [0..0] of TVmtMethodEntry;
end

```

TVmtMethodTable is a record structure describing all the VMT methods of a class. It is basically an array of pointers TVmtMethodTable.Entry ([2149](#)).

78.25.1 Property overview

Page	Properties	Access	Description
2149	Entry	r	Indexed access to VMT method information.

78.25.2 TVmtMethodTable.Entry

Synopsis: Indexed access to VMT method information.

Declaration: Property Entry[Index: LongWord]: PVmtMethodEntry

Visibility: public

Access: Read

Description: Method offers indexed access to the method descriptions. It returns the location of the Index-th method of the VMT table Index must be in the range 0 to Count-1 (included).

See also: PVmtMethodEntry ([2075](#)), TVmtMethodEntry ([2081](#))

78.26 EPropertyConvertError

78.26.1 Description

EPropertyConvertError is not used in the Free Pascal implementation of the typinfo unit, but is declared for Delphi compatibility.

78.27 EPropertyError

78.27.1 Description

Exception raised in case of an error in one of the functions.

Chapter 79

Reference for unit 'unicodedata'

79.1 Used units

Table 79.1: Used units by unit 'unicodedata'

Name	Page
System	1335

79.2 Overview

The `fpwiderstring` ([2150](#)) unit relies on having relevant Unicode collation data linked in the binary. The Unicode data is managed using the routines in the `unicodedata` unit. The FPC project distributes some Unicode collation data in `.bco` files which can be loaded using the `LoadCollation` ([2161](#)) routines. The `LoadCollation` is the main routine of this unit.

All collation data requires at least the Default Unicode Collation Element Table to be registered (called `DUCET`). The `DUCET` encoding is provided by the `unicodeducet` unit, part of the `rtl-Unicode` package.

There are two ways to register collations :

1. **at compile time**: by including the desired collation unit, for example for Russian and Japanese languages to be available you will have to include `collation_ru` and `collation_ja` from package "rtl-unicode".
2. **at runtime** using the above mentioned `LoadCollation` function.

The two ways can co-exist: some collations may be compile time included (for example for most used collations) and others can be loaded at runtime in the same application.

The binary collation files are endian sensitive:

- there are files for little endian systems named `collation__le.bco` (such as `collation_ru_le.bco` and `collation_ja_le.bco`)
- there are files for big endian systems named `collation__be.bco` (such as `collation_ru_be.bco` and `collation_ja_be.bco`).

Note that the compile time units collation units (collation_**lang**.pas) include already the unicode-ducet.pas (DUCET) unit so it is not necessary to include it manually, contrary to the binary files. So an application that only uses the binary collation files should at least include the `unicodeducet` unit or manually load the binary collation `collation_ducet_le.bco` or `collation_ducet_be.bco`, depending on the endianness of the platform. The `LoadCollation` (2161) call using a directory and the language `ducet` automatically select the correct file.

79.3 Constants, types and variables

79.3.1 Resource strings

```
SCollationNotFound = 'Collation not found : "%s".'
```

Error message when a collation is not found.

79.3.2 Constants

```
DEFAULT_UCA_COMPARISON_STRENGTH = 3
```

```
DirectorySeparator = '/'
```

```
ENDIAN_SUFFIX : Array[TEndianKind] of UnicodeString = ('le', 'be'
)
```

`ENDIAN_SUFFIX` contains the suffixes used in `LoadCollation` (2161) when constructing a collation filename for a language.

```
ERROR_INVALID_CODEPOINT_SEQUENCE = 1
```

Error exit code for `UnicodeToLower/UnicodeToUpper`.

```
HIGH_SURROGATE_BEGIN = Word($D800)
```

First value for high surrogate values.

```
HIGH_SURROGATE_COUNT = HIGH_SURROGATE_END - HIGH_SURROGATE_BEGIN
+ 1
```

Number of high surrogate values.

```
HIGH_SURROGATE_END = Word($DBFF)
```

Last value for high surrogate values.

```
LOW_SURROGATE_BEGIN = Word($DC00)
```

First value for low surrogate values.

```
LOW_SURROGATE_END = Word($DFFF)
```


Last value for low surrogate values.

`MAX_LEGAL_UTF32 = $10FFFF`

Maximum value of a legally encoded UTF32 value (currently unused).

`MAX_WORD = High (Word)`

`MAX_WORD` is the maximum value of a `WORD` typed value.

`reCodesetConversion = reRangeError`

`ROOT_COLLATION_NAME = 'DUCET'`

Name of the root collation (used if no collation name is given).

`UCS4_HALF_BASE = LongWord($10000)`

Offset for UCS4 character encoding.

`UCS4_HALF_MASK = Word($3FF)`

Unused currently.

`UGC_ClosePunctuation = 14`

Unicode token category: Close punctuation.

`UGC_CombiningMark = 6`

Unicode token category: Uppercase letter.

`UGC_ConnectPunctuation = 11`

Unicode token category: Connect punctuation.

`UGC_Control = 25`

Unicode token category: control token.

`UGC_CurrencySymbol = 19`

Unicode token category: Currency symbol.

`UGC_DashPunctuation = 12`

Unicode token category: Dash punctuation.

`UGC_DecimalNumber = 8`

Unicode token category: Uppercase letter.

UGC_EnclosingMark = 7

Unicode token category: Uppercase letter.

UGC_FinalPunctuation = 16

Unicode token category: Final punctuation.

UGC_Format = 26

Unicode token category: Formatting token.

UGC_InitialPunctuation = 15

Unicode token category: Initial punctuation.

UGC_LetterNumber = 9

Unicode token category: Letter-number.

UGC_LineSeparator = 23

Unicode token category: Line separator.

UGC_LowercaseLetter = 1

Unicode general category: Lowercase letter.

UGC_MathSymbol = 18

Unicode token category: Math symbol.

UGC_ModifierLetter = 3

Unicode general category: modifier letter.

UGC_ModifierSymbol = 20

Unicode token category: modifier symbol.

UGC_NonSpacingMark = 5

Unicode general category: Non-spacing mark.

UGC_OpenPunctuation = 13

Unicode token category: Open punctuation.

UGC_OtherLetter = 4

Unicode general category: Other letter.

UGC_OtherNumber = 10

Unicode token category: Other number.

UGC_OtherPunctuation = 17

Unicode token category: Other punctuation.

UGC_OtherSymbol = 21

Unicode token category: Other symbol.

UGC_ParagraphSeparator = 24

Unicode token category: Paragraph separator.

UGC_PrivateUse = 28

Unicode token category: For private use.

UGC_SpaceSeparator = 22

Unicode token category: Space separator.

UGC_Surrogate = 27

Unicode token category: Surrogate token.

UGC_TitlecaseLetter = 2

Unicode general category: Titlecase letter.

UGC_Unassigned = 29

Unicode token category: As yet unassigned.

UGC_UppercaseLetter = 0

Unicode general category: Uppercase letter.

```
UnicodeCategoryNames : Array[0..29] of string = ('Lu', 'Ll', 'Lt'
  , 'Lm', 'Lo', 'Mn', 'Mc', 'Me', 'Nd', 'Nl', 'No', 'Pc', 'Pd', 'Ps'
  , 'Pe', 'Pi', 'Pf', 'Po', 'Sm', 'Sc', 'Sk', 'So', 'Zs', 'Zl', 'Zp'
  , 'Cc', 'Cf', 'Cs', 'Co', 'Cn')
```

ZERO_UINT24 : UInt24 = (c: 0; b: 0; a: 0)

TUInt24Rec value with all bytes zero.

79.3.3 Types

PCollationTableItem = ^TCollationTableItem

PUCA_DataBook = ^TUCA_DataBook

Pointer to TUCA_DataBook type.

PUCA_PropItemContextRec = ^TUCA_PropItemContextRec

Pointer to TUCA_PropItemContextRec.

PUCA_PropItemContextTreeNodeRec =
^TUCA_PropItemContextTreeNodeRec

Pointer to TUCA_PropItemContextTreeNodeRec.

PUCA_PropItemContextTreeRec = ^TUCA_PropItemContextTreeRec

Pointer to TUCA_PropItemContextTreeRec.

PUCA_PropItemRec = ^TUCA_PropItemRec

Pointer to TUCA_PropItemRec.

PUCA_PropWeights = ^TUCA_PropWeights

Pointer to TUCA_PropWeights.

PUC_Prop = ^TUC_Prop

Pointer to TUC_Prop record.

PUInt24 = ^UInt24

Pointer to TUInt24Rec.

TCategoryMask = Set of

TCollationField = (BackWard, VariableLowLimit, VariableHighLimit,
Alternate, Normalization, Strength)

Table 79.2: Enumeration values for type TCollationField

Value	Explanation
Alternate	
BackWard	Backwards encoded.
Normalization	
Strength	
VariableHighLimit	Has upper bound on variable weights.
VariableLowLimit	Has lower bound on variable weights.

TCollationField describes some properties of the collation data items.

TCollationFields = Set of TCollationField

Set of TCollationField.

TCollationName = Array[0..(128-1)] of Byte

Collation name string type (fixed length).

```
TCollationTableItem = record
public
  Collation : PUCA_DataBook;
  Aliases : TUnicodeStringArray;
end
```

TCollationTableItemArray = Array of TCollationTableItem

TCollationVersion = TCollationName

TEndianKind = (Little,Big)

Table 79.3: Enumeration values for type TEndianKind

Value	Explanation
Big	Big-endian platform.
Little	Little-endian platform.

TEndianKind is an auxiliary enumerated type to enumerate the endianness of platforms.

TSetOfByte = Set of Byte

TUCASortKey = Array of TUCASortKeyItem

Array of TUCASortKeyItem.

TUCASortKeyItem = Word

Alias for WORD.

```
TUCA_PropWeights = packed record
public
  Weights : Array[0..2] of
    Word;
end
```

TUC_PropWeights describes the weights of collation characteristics of a unicode character. It is an internal structure which should not be used directly, the actual structure is subject to change.

```
TUCA_VariableKind = (ucaShifted, ucaNonIgnorable, ucaBlanked,
    ucaShiftedTrimmed, ucaIgnoreSP)
```

Table 79.4: Enumeration values for type TUCA_VariableKind

Value	Explanation
ucaBlanked	Variable collation elements and any subsequent ignorable collation elements are reset so that all weigh
ucaIgnoreSP	Not implemented (variant of Shifted that reduces the set of variable collation elements to include only
ucaNonIgnorable	Variable collation elements are not reset to be quaternary collation elements.
ucaShifted	Variable collation elements are reset to zero at levels one through three.
ucaShiftedTrimmed	This option is the same as Shifted, except that all trailing FFFFs are trimmed from the sort key.

Options for weighting data.

```
TUnicodeStringArray = Array of UnicodeString
```

```
UInt24 = TUInt24Rec
```

Alias for TUInt24Rec.

79.4 Procedures and functions

79.4.1 AddAliasCollation

```
Declaration: function AddAliasCollation(ACollation: PUCA_DataBook;
    AALias: UnicodeString) : Boolean
```

Visibility: default

79.4.2 BytesToName

```
Declaration: function BytesToName(const ABytes: Array of Byte) : UnicodeString
```

Visibility: default

79.4.3 BytesToString

```
Declaration: function BytesToString(const ABytes: Array of Byte;
    const AValideChars: TSetOfByte) : UnicodeString
```

Visibility: default

79.4.4 CanonicalOrder

Synopsis: Put unicode string in canonical order.

```
Declaration: procedure CanonicalOrder(var AString: UnicodeString); Overload
    procedure CanonicalOrder(AStr: PUnicodeChar; const ALength: SizeInt)
        ; Overload
```

Visibility: default

Description: CanonicalOrder transforms a unicode string AString (or the alternate form using a null-terminated AStr with length ALength) so it is in canonical order (as defined by the unicode specification). A string needs to be in canonical order to be able to compare strings. This function is called as part of NormalizeNFD (2162).

See also: NormalizeNFD (2162)

79.4.5 CompareSortKey

Synopsis: Compare two sort keys.

Declaration:

```
function CompareSortKey(const A: TUCASortKey; const B: TUCASortKey)
    : Integer; Overload
function CompareSortKey(const A: TUCASortKey; const B: Array of Word)
    : Integer; Overload
```

Visibility: default

Description: CompareSortKey compares 2 sort keys A and B. It returns

- a negative number if A comes alphabetically ordered before B.
- Zero if A is alphabetically identical to B.
- A positive number if A is alphabetically ordered after B.

Sort keys can be constructed from unicode strings using ComputeSortKey (2158).

See also: ComputeSortKey (2158)

79.4.6 ComputeSortKey

Synopsis: Compute the sort key for a string.

Declaration:

```
function ComputeSortKey(const AString: UnicodeString;
    const ACollation: PUCA_DataBook) : TUCASortKey
    ; Overload
function ComputeSortKey(const AStr: PUnicodeChar;
    const ALength: SizeInt;
    const ACollation: PUCA_DataBook) : TUCASortKey
    ; Overload
```

Visibility: default

Description: ComputeSortKey computes the sort key for a unicode string AString (or the alternate form using a null-terminated AStr with length ALength) using the Unicode Collation Algorithm data in ACollation. This key can then be used in CompareSortKey (2158) to compare unicode strings.

See also: CompareSortKey (2158)

79.4.7 FilterString

Declaration: `function FilterString(const AStr: PUnicodeChar; const ALength: SizeInt;
const AExcludedMask: TCategoryMask) : UnicodeString
; Overload
function FilterString(const AStr: UnicodeString;
const AExcludedMask: TCategoryMask) : UnicodeString
; Overload`

Visibility: default

79.4.8 FindCollation

Synopsis: Find a collation by name.

Declaration: `function FindCollation(AName: UnicodeString) : PUCA_DataBook; Overload
function FindCollation(const AIndex: Integer) : PUCA_DataBook; Overload`

Visibility: default

Description: `FindCollection` searches a collation with name `AName` or index `AIndex` in the list of known collations and returns a pointer to the collation data. If the requested collation is not known, i.e. the name is not found, or the index is out of range, then `Nil` is returned. The valid index range is 0 to `GetCollationCount-1`.

See also: `GetCollationCount` ([2160](#))

79.4.9 FreeCollation

Synopsis: Free collation data.

Declaration: `procedure FreeCollation(AItem: PUCA_DataBook)`

Visibility: default

Description: `FreeCollation` removes all structures in the collation data from memory. It will not do anything when the header field `Dynamic` is `false`. (collations loaded and registered using `LoadCollation` ([2161](#)) are always dynamic and must be freed).

See also: `LoadCollation` ([2161](#))

79.4.10 FromUCS4

Synopsis: Convert UCS4 to `UnicodeChar`.

Declaration: `procedure FromUCS4(const AValue: UCS4Char; out AHighS: UnicodeChar;
out ALowS: UnicodeChar)`

Visibility: default

Description: `FromUCS4` converts the UCS4 encoded unicode character `AValue` to a set of unicode (surrogate pair) characters encoded in UTF16: `AHighS`, `ALowS`.

See also: `ToUCS4` ([2163](#)), `UnicodeIsHighSurrogate` ([2163](#)), `UnicodeIsLowSurrogate` ([2163](#)), `UnicodeIsSurrogatePair` ([2164](#))

Declaration:

```
function IncrementalCompareString(const AStrA: PUnicodeChar;
                                const ALengthA: SizeInt;
                                const AStrB: PUnicodeChar;
                                const ALengthB: SizeInt;
                                const ACollation: PUCA_DataBook)
                                : Integer; Overload
function IncrementalCompareString(const AStrA: UnicodeString;
                                const AStrB: UnicodeString;
                                const ACollation: PUCA_DataBook)
                                : Integer; Overload
```

Visibility: default

Description: `IncrementalCompareString` creates 2 compare keys from the strings `AStrA` and `AStrB` using collation data in `ACollation`. The keys are computed only to the point where the two strings differ. This means the keys cannot be reused for other comparisons if the strings differ. The two strings can be specified as a unicode string or as a pointer to a null-terminated character array with a length (`ALengthA` and `ALengthB`). It returns then the result of `CompareSortKey` (2158).

Errors: None.

See also: `ComputeSortKey` (2158), `CompareSortKey` (2158)

79.4.15 LoadCollation

Synopsis: Load a binary collation data file from file.

Declaration:

```
function LoadCollation(const AData: Pointer;
                      const ADataLength: Integer;
                      var AAliases: TUnicodeStringArray) : PUCA_DataBook
                      ; Overload
function LoadCollation(const AData: Pointer; const ADataLength: Integer)
                      : PUCA_DataBook; Overload
function LoadCollation(const AFileName: UnicodeString;
                      var AAliases: TUnicodeStringArray) : PUCA_DataBook
                      ; Overload
function LoadCollation(const AFileName: UnicodeString) : PUCA_DataBook
                      ; Overload
function LoadCollation(const ADirectory: UnicodeString;
                      const ALanguage: UnicodeString;
                      var AAliases: TUnicodeStringArray) : PUCA_DataBook
                      ; Overload
function LoadCollation(const ADirectory: UnicodeString;
                      const ALanguage: UnicodeString) : PUCA_DataBook
                      ; Overload
```

Visibility: default

Description: `LoadCollation` loads collation data from file `AFileName`, or from a memory block `AData` with length `ADataLength`. If successful, it returns a pointer which can be used to register the collation using `RegisterCollation` (2162). if there is a problem with the data, `Nil` is returned.

The filename can also be specified as a `Directory` and language name `ALanguage`. The latter is prepended with `collation_` and appended with the native endianness of the current platform, and has extension `.bco`

Errors: If the file containing data does not exist or has a size which is less than the encoded header size, `Nil` is returned.

See also: [RegisterCollation \(2162\)](#)

79.4.16 NormalizeNFD

Synopsis: Perform unicode normalization D on a string.

Declaration:

```
function NormalizeNFD(const AString: UnicodeString) : UnicodeString
    ; Overload
function NormalizeNFD(const AStr: PUnicodeChar; ALength: SizeInt)
    : UnicodeString; Overload
```

Visibility: default

Description: `NormalizeNFD` normalizes the string `AString` (or the alternate form using a null-terminated `AStr` with length `ALength`) to Unicode Normalization Form D. The resulting string can be used to determine equivalence of unicode strings.

See also: [CanonicalOrder \(2157\)](#)

79.4.17 PrepareCollation

Synopsis: Prepare a collation for use in the list.

Declaration:

```
procedure PrepareCollation(ACollation: PUCA_DataBook;
    const ABaseName: UnicodeString;
    const AChangedFields: TCollationFields)
```

Visibility: default

Description: `PrepareCollation` will link collation definition `ACollation` to the base collection with name `ABaseName` (if empty, it defaults to the root collation). It will also initialize some fields in the definition, copying them from the base collation, but excludes the fields enumerated in `AChangedFields`. It should normally not be needed to call this function, it is called as part of `LoadCollation` ([2161](#)).

See also: [LoadCollation \(2161\)](#)

79.4.18 RegisterCollation

Synopsis: Register a new collation.

Declaration:

```
function RegisterCollation(const ACollation: PUCA_DataBook) : Boolean
    ; Overload
function RegisterCollation(const ACollation: PUCA_DataBook;
    const AAliasList: Array of UnicodeString)
    : Boolean; Overload
function RegisterCollation(ADirectory: UnicodeString;
    ALanguage: UnicodeString) : Boolean; Overload
```

Visibility: default

Description: `RegisterCollation` registers a new collation `ACollation` in the list of known collations. The collation data can be specified directly (`ACollation`) or a name of a language (`ALanguage`) in a directory (`ADirectory`). The latter option will load the binary encoded collation from file using `LoadCollation` (2161).

If the collation is loaded correctly, `True` is returned, otherwise `False` is returned (for instance when a collation with the same name is already loaded).

See also: `UnRegisterCollation` (2165), `UnRegisterCollations` (2165), `LoadCollation` (2161)

79.4.19 ToUCS4

Synopsis: Encode unicode UTF16 surrogate pair to UCS4 character.

Declaration: `function ToUCS4(const AHighS: UnicodeChar; const ALowS: UnicodeChar) : UCS4Char`

Visibility: default

Description: `ToUCS4` converts set of unicode (surrogate pair) characters encoded in UTF16: `AHighS`, `ALowS` to a UCS4 encoded unicode character.

See also: `FromUCS4` (2159), `UnicodeIsHighSurrogate` (2163), `UnicodeIsLowSurrogate` (2163), `UnicodeIsSurrogatePair` (2164)

79.4.20 UnicodeIsHighSurrogate

Synopsis: Check if a UTF16 character is the high character in a surrogate pair.

Declaration: `function UnicodeIsHighSurrogate(const AValue: UnicodeChar) : Boolean`

Visibility: default

Description: `UnicodeIsHighSurrogate` checks whether `AValue` is a valid high character of a surrogate pair, and returns `True` if this is the case. It does this by checking whether the values are within the bounds for high characters in surrogate pairs.

See also: `FromUCS4` (2159), `ToUCS4` (2163), `UnicodeIsHighSurrogate` (2163), `UnicodeIsLowSurrogate` (2163), `UnicodeIsSurrogatePair` (2164)

79.4.21 UnicodeIsLowSurrogate

Synopsis: Check if a UTF16 character is the low character in a surrogate pair.

Declaration: `function UnicodeIsLowSurrogate(const AValue: UnicodeChar) : Boolean`

Visibility: default

Description: `UnicodeIsLowSurrogate` checks whether `AValue` is a valid low character of a surrogate pair, and returns `True` if this is the case. It does this by checking whether the values are within the bounds for low characters in surrogate pairs.

See also: `FromUCS4` (2159), `ToUCS4` (2163), `UnicodeIsHighSurrogate` (2163), `UnicodeIsSurrogatePair` (2164)

79.4.22 `UnicodeIsSurrogatePair`

Synopsis: Check if a pair of UTF16 encoded characters is a valid surrogate pair.

Declaration:

```
function UnicodeIsSurrogatePair(const AHighSurrogate: UnicodeChar;
                                const ALowSurrogate: UnicodeChar)
                                : Boolean
```

Visibility: default

Description: `UnicodeIsSurrogatePair` checks whether `AHighSurrogate`, `ALowSurrogate` constitute a valid surrogate pair, and returns `True` if this is the case. It does this by checking whether the values are within the bounds for high and low surrogate pairs.

See also: `FromUCS4` (2159), `ToUCS4` (2163), `UnicodeIsHighSurrogate` (2163), `UnicodeIsLowSurrogate` (2163)

79.4.23 `UnicodeToLower`

Synopsis: Transform unicode string to lowercase.

Declaration:

```
function UnicodeToLower(const AString: UnicodeString;
                        const AIgnoreInvalidSequence: Boolean;
                        out AResultString: UnicodeString) : Integer
```

Visibility: default

Description: `UnicodeToLower` transforms a UTF16 unicode string `AString` to its lowercase equivalent and returns this in `AResultString`. If the transformation was successful, then the function returns 0. A nonzero return value means an error occurred. `AResultString` will remain untouched in that case.

If a character in `AString` cannot be found in the unicode data tables, an error will be reported, unless `AIgnoreInvalidSequence` is set to `True`, in which case the character will be copied as-is to the output.

Unicode collation data can be loaded using `RegisterCollation` (2162) or `LoadCollation` (2161)

Errors: On error, a nonzero value will be returned.

See also: `UnicodeToUpper` (2164), `RegisterCollation` (2162), `LoadCollation` (2161)

79.4.24 `UnicodeToUpper`

Synopsis: Transform unicode string to uppercase.

Declaration:

```
function UnicodeToUpper(const AString: UnicodeString;
                        const AIgnoreInvalidSequence: Boolean;
                        out AResultString: UnicodeString) : Integer
```

Visibility: default

Description: `UnicodeToUpper` transforms a UTF16 unicode string `AString` to its uppercase equivalent and returns this in `AResultString`. If the transformation was successful, then the function returns 0. A nonzero return value means an error occurred. `AResultString` will remain untouched in that case.

If a character in `AString` cannot be found in the unicode data tables, an error will be reported, unless `AIgnoreInvalidSequence` is set to `True`, in which case the character will be copied as-is to the output.

Unicode collation data can be loaded using `RegisterCollation` (2162) or `LoadCollation` (2161)

Errors: On error, a nonzero value will be returned.

See also: [UnicodeToLower \(2164\)](#), [RegisterCollation \(2162\)](#), [LoadCollation \(2161\)](#)

79.4.25 UnregisterCollation

Synopsis: Remove a collation from the list of known collections.

Declaration: `function UnregisterCollation(AName: UnicodeString) : Boolean`

Visibility: default

Description: `UnRegisterCollation` removes a collation `AName` from the list of known collations. It returns `True` if the collation was found and successfully removed.

Errors: If the collation was not found, `False` is returned.

See also: [RegisterCollation \(2162\)](#), [UnRegisterCollations \(2165\)](#), [LoadCollation \(2161\)](#)

79.4.26 UnregisterCollations

Synopsis: Unregister all collations.

Declaration: `procedure UnregisterCollations(const AFreeDynamicCollations: Boolean)`

Visibility: default

Description: `UnregisterCollations` unregisters all known collations. If `AFreeDynamicCollations` is `True`, then dynamic collations will be removed from memory using [FreeCollation \(2159\)](#). This must normally be set to `true`.

See also: [RegisterCollation \(2162\)](#), [UnRegisterCollation \(2165\)](#), [LoadCollation \(2161\)](#), [FreeCollation \(2159\)](#)

79.5 TCollationTable

```
TCollationTable = record
private
    FItems : TCollationTableItemArray
    ;
    FCount : Integer;
    function GetCapacity : Integer;
    function
    GetCount : Integer;
    function GetItem(const AIndex: Integer) : PCollationTableItem
    ;
    procedure Grow;
    procedure ClearItem(AItem: PCollationTableItem
    );
    AddAlias;
public
    class function NormalizeName(AName: UnicodeString
    ) : UnicodeString
    ; Static;

    procedure
    Clear;
```

```

    IndexOf;
    Find;
    function Add(ACollation: PUCA_DataBook
    ) : Integer;
    function Remove(AIndex: Integer) : PUCA_DataBook;
    property Item[AIndex: Integer]: PCollationTableItem; default;
    property Count : Integer;
    property Capacity : Integer;
end

```

79.5.1 Method overview

Page	Method	Description
2167	Add	
2166	Clear	
2166	Find	
2166	IndexOf	
2166	NormalizeName	
2167	Remove	

79.5.2 Property overview

Page	Properties	Access	Description
2167	Capacity	r	
2167	Count	r	
2167	Item	r	

79.5.3 TCollationTable.NormalizeName

Declaration: `class function NormalizeName(AName: UnicodeString) : UnicodeString; Static`

Visibility: public

79.5.4 TCollationTable.Clear

Declaration: `procedure Clear`

Visibility: public

79.5.5 TCollationTable.IndexOf

Declaration: `function IndexOf(AName: UnicodeString) : Integer; Overload`
`function IndexOf(ACollation: PUCA_DataBook) : Integer; Overload`

Visibility: public

79.5.6 TCollationTable.Find

Declaration: `function Find(AName: UnicodeString) : PCollationTableItem; Overload`
`function Find(ACollation: PUCA_DataBook) : PCollationTableItem`
`; Overload`

Visibility: public

79.5.7 TCollationTable.Add

Declaration: function Add(ACollation: PUCA_DataBook) : Integer

Visibility: public

79.5.8 TCollationTable.Remove

Declaration: function Remove(AIndex: Integer) : PUCA_DataBook

Visibility: public

79.5.9 TCollationTable.Item

Declaration: Property Item[AIndex: Integer]: PCollationTableItem; default

Visibility: public

Access: Read

79.5.10 TCollationTable.Count

Declaration: Property Count : Integer

Visibility: public

Access: Read

79.5.11 TCollationTable.Capacity

Declaration: Property Capacity : Integer

Visibility: public

Access: Read

79.6 TUCA_DataBook

```
TUCA_DataBook = record
public
  Base : PUCA_DataBook;
  Version :
    TCollationVersion;
  CollationName : TCollationName;
  VariableWeight
    : TUCA_VariableKind;
  Backwards : Array[0..3] of Boolean;
  BMP_Table1
    : PByte;
  BMP_Table2 : PUInt24;
```



```

OBMP_Table1 : PWord;
OBMP_Table2
: PUInt24;
PropCount : Integer;
Props : PUCA_PropItemRec;
VariableLowLimit
: Word;
VariableHighLimit : Word;
NoNormalization : Boolean;
ComparisonStrength : Byte;
Dynamic : Boolean;
function IsVariable
(const AWeight: PUCA_PropWeights) : Boolean;
end

```

TUCA_DataBook describes a Unicode Collation Algorithm data set. data sets can be registered using the RegisterCollation (2162) function or loaded from file using LoadCollation (2161). A collation data book must be specified when comparing unicode strings.

79.6.1 Method overview

Page	Method	Description
2168	IsVariable	Check if a weight is a variable weight.

79.6.2 TUCA_DataBook.IsVariable

Synopsis: Check if a weight is a variable weight.

Declaration: function IsVariable(const AWeight: PUCA_PropWeights) : Boolean

Visibility: public

Description: IsVariable checks whether AWeight is between the VariableLowLimit and VariableHighLimit limits.

79.7 TUCA_PropItemContextRec

```

TUCA_PropItemContextRec = packed record
public
    CodePointCount :
    Byte;
    WeightCount : Byte;
    function GetCodePoints : PUInt24;
    function GetWeights : PUCA_PropWeights;
end

```

This is an internal structure which should not be used directly, the actual structure is subject to change.

79.7.1 Method overview

Page	Method	Description
2169	GetCodePoints	get the address of actual code points.
2169	GetWeights	Get the Address of actual weights.

79.7.2 TUCA_PropltemContextRec.GetCodePoints

Synopsis: get the address of actual code points.

Declaration: `function GetCodePoints : PUInt24`

Visibility: public

79.7.3 TUCA_PropltemContextRec.GetWeights

Synopsis: Get the Address of actual weights.

Declaration: `function GetWeights : PUCA_PropWeights`

Visibility: public

79.8 TUCA_PropItemContextTreeNodeRec

```
TUCA_PropItemContextTreeNodeRec = packed record
public
    Left : Word
    ;
    Right : Word;
    Data : TUCA_PropItemContextRec;
    function GetLeftNode
    : PUCA_PropItemContextTreeNodeRec;
    function GetRightNode : PUCA_PropItemContextTreeNodeRec
    ;
end
```

This is an internal structure for the tree which should not be used directly, the actual structure is subject to change.

79.8.1 Method overview

Page	Method	Description
2169	GetLeftNode	Access to left tree node data.
2170	GetRightNode	Access to right tree node data.

79.8.2 TUCA_PropItemContextTreeNodeRec.GetLeftNode

Synopsis: Access to left tree node data.

Declaration: `function GetLeftNode : PUCA_PropItemContextTreeNodeRec`

Visibility: public

79.8.3 TUCA_PropltemContextTreeNodeRec.GetRightNode

Synopsis: Access to right tree node data.

Declaration: `function GetRightNode : PUCA_PropltemContextTreeNodeRec`

Visibility: public

79.9 TUCA_PropltemContextTreeRec

```
TUCA_PropltemContextTreeRec = packed record
public
  Size : UInt24
  ;
  function GetData : PUCA_PropltemContextTreeNodeRec;
  property
    Data : PUCA_PropltemContextTreeNodeRec;
  function Find(const AChars
    : PUInt24; const ACharCount: Integer;
    out ANode: PUCA_PropltemContextTreeNodeRec
  ) : Boolean;
end
```

This is an internal tree structure for storing unicode collation data which should not be used directly, the actual structure is subject to change.

79.9.1 Method overview

Page	Method	Description
2170	Find	Find data for encoded character.
2170	GetData	Access to tree data (getter for Data property).

79.9.2 Property overview

Page	Properties	Access	Description
2171	Data	r	Read-only access to tree data.

79.9.3 TUCA_PropltemContextTreeRec.GetData

Synopsis: Access to tree data (getter for Data property).

Declaration: `function GetData : PUCA_PropltemContextTreeNodeRec`

Visibility: public

79.9.4 TUCA_PropltemContextTreeRec.Find

Synopsis: Find data for encoded character.

Declaration: `function Find(const AChars: PUInt24; const ACharCount: Integer;
 out ANode: PUCA_PropltemContextTreeNodeRec) : Boolean`

Visibility: public

Description: Find searches the tree for the collation data for the character encoded in AChars (ACharCount). It returns true if the data was found, false if not. A pointer to the collation data is returned in ANode.

79.9.5 TUCA_PropItemContextTreeRec.Data

Synopsis: Read-only access to tree data.

Declaration: Property Data : PUCA_PropItemContextTreeNodeRec

Visibility: public

Access: Read

79.10 TUCA_PropItemRec

```
TUCA_PropItemRec = packed record
private
  FLAG_VALID = 0;
  FLAG_CODEPOINT
    = 1;
  FLAG_CONTEXTUAL = 2;
  FLAG_DELETION = 3;
  FLAG_COMPRESS_WEIGHT_1
    = 6;
  FLAG_COMPRESS_WEIGHT_2 = 7;
  function GetCodePoint : UInt24
    ;
public
  WeightLength : Byte;
  ChildCount : Byte;
  Size : Word
    ;
  Flags : Byte;
  function HasCodePoint : Boolean;
  property CodePoint
    : UInt24;
  function IsValid : Boolean;
  procedure GetWeightArray
    (ADest: PUCA_PropWeights);
  function GetSelfOnlySize : Cardinal;
  function GetContextual : Boolean;
  property Contextual : Boolean
    ;
  function GetContext : PUCA_PropItemContextTreeRec;
  function
    IsDeleted : Boolean;
  function IsWeightCompress_1 : Boolean;
  function
    IsWeightCompress_2 : Boolean;
end
```

TUCA_PropItemRec encodes 1 entry from the Unicode Collation data in an encoded form.

79.10.1 Method overview

Page	Method	Description
2173	<code>GetContext</code>	Access to context data.
2173	<code>GetContextual</code>	Check if the contextual bit is set in the flags (getter for Contextual).
2172	<code>GetSelfOnlySize</code>	Size of this item data (in bytes).
2172	<code>GetWeightArray</code>	Return an array of weights.
2172	<code>HasCodePoint</code>	Check flags whether a codepoint is present.
2173	<code>IsDeleted</code>	Check flags if deleted bit is set.
2172	<code>IsValid</code>	Check flags for validity.
2173	<code>IsWeightCompress_1</code>	Check whether weight compression flag 1 is set.
2173	<code>IsWeightCompress_2</code>	Check whether weight compression flag 2 is set.

79.10.2 Property overview

Page	Properties	Access	Description
2173	<code>CodePoint</code>	r	Get the codepoint.
2174	<code>Contextual</code>	r	Check if the contextual bit is set in the flags.

79.10.3 TUCA_ProplItemRec.HasCodePoint

Synopsis: Check flags whether a codepoint is present.

Declaration: `function HasCodePoint : Boolean`

Visibility: public

79.10.4 TUCA_ProplItemRec.IsValid

Synopsis: Check flags for validity.

Declaration: `function IsValid : Boolean`

Visibility: public

79.10.5 TUCA_ProplItemRec.GetWeightArray

Synopsis: Return an array of weights.

Declaration: `procedure GetWeightArray (ADest: PUCA_PropWeights)`

Visibility: public

Description: `GetWeightArray` returns an array with the weights in `ADest`. `ADest` must point to enough room for `WeightLength` weights.

79.10.6 TUCA_ProplItemRec.GetSelfOnlySize

Synopsis: Size of this item data (in bytes).

Declaration: `function GetSelfOnlySize : Cardinal`

Visibility: public

Description: Return the size of the item's data and properties.

79.10.7 TUCA_PropltemRec.GetContextual

Synopsis: Check if the contextual bit is set in the flags (getter for Contextual).

Declaration: `function GetContextual : Boolean`

Visibility: public

79.10.8 TUCA_PropltemRec.GetContext

Synopsis: Access to context data.

Declaration: `function GetContext : PUCA_PropItemContextTreeRec`

Visibility: public

Description: `GetContext` returns a pointer to the context data. It is `Nil` if the context flag is not set.

79.10.9 TUCA_PropltemRec.IsDeleted

Synopsis: Check flags if deleted bit is set.

Declaration: `function IsDeleted : Boolean`

Visibility: public

79.10.10 TUCA_PropltemRec.IsWeightCompress_1

Synopsis: Check whether weight compression flag 1 is set.

Declaration: `function IsWeightCompress_1 : Boolean`

Visibility: public

79.10.11 TUCA_PropltemRec.IsWeightCompress_2

Synopsis: Check whether weight compression flag 2 is set.

Declaration: `function IsWeightCompress_2 : Boolean`

Visibility: public

79.10.12 TUCA_PropltemRec.CodePoint

Synopsis: Get the codepoint.

Declaration: `Property CodePoint : UInt24`

Visibility: public

Access: Read

Description: Access to codepoint if `HasCodePoint` returns true. If `HasCodePoint` returns false, an exception will be raised.

79.10.13 TUCA_PropltemRec.Contextual

Synopsis: Check if the contextual bit is set in the flags.

Declaration: Property Contextual : Boolean

Visibility: public

Access: Read

79.11 TUC_Prop

```
TUC_Prop = packed record
private
    function GetCategory : Byte;
    procedure SetCategory(AValue: Byte);
    function GetWhiteSpace : Boolean
    ;
    procedure SetWhiteSpace(AValue: Boolean);
    function GetHangulSyllable
    : Boolean;
    procedure SetHangulSyllable(AValue: Boolean);
    function
    GetNumericValue : Double;
    function GetUnifiedIdeograph : Boolean
    ;
public
    C : Byte;
    C3 : Byte;
    N : Byte;
    UC : UInt24;
    LC
    : UInt24;
    D : SmallInt;
    property CategoryData : Byte;
    property
    NumericIndex : Byte;
    property SimpleUpperCase : UInt24;
    property
    SimpleLowerCase : UInt24;
    property DecompositionID : SmallInt;
    property Category : Byte;
    property WhiteSpace : Boolean;
    property
    HangulSyllable : Boolean;
    property UnifiedIdeograph : Boolean;
    property NumericValue : Double;
end
```

TUC_Prop describes the collation characteristics of a unicode character. It is an internal structure which should not be used directly, the actual structure is subject to change.

79.11.1 Property overview

Page	Properties	Access	Description
2176	Category	rw	Get the category.
2175	CategoryData	rw	Category data.
2176	DecompositionID	rw	Decomposition ID.
2176	HangulSyllable	rw	Is the character a hangul syllable.
2175	NumericIndex	rw	Index in numeric values array.
2177	NumericValue	r	Numeric value.
2175	SimpleLowerCase	rw	Simple lowercase version.
2175	SimpleUpperCase	rw	Simple uppercase version.
2176	UnifiedIdeograph	r	
2176	WhiteSpace	rw	Is the character considered whitespace.

79.11.2 TUC_Prop.CategoryData

Synopsis: Category data.

Declaration: `Property CategoryData : Byte`

Visibility: public

Access: Read,Write

Description: `CategoryData` is a bitmask encoding various characteristics of the unicode character.

79.11.3 TUC_Prop.NumericIndex

Synopsis: Index in numeric values array.

Declaration: `Property NumericIndex : Byte`

Visibility: public

Access: Read,Write

79.11.4 TUC_Prop.SimpleUpperCase

Synopsis: Simple uppercase version.

Declaration: `Property SimpleUpperCase : UInt24`

Visibility: public

Access: Read,Write

79.11.5 TUC_Prop.SimpleLowerCase

Synopsis: Simple lowercase version.

Declaration: `Property SimpleLowerCase : UInt24`

Visibility: public

Access: Read,Write

79.11.6 TUC_Prop.DecompositionID

Synopsis: Decomposition ID.

Declaration: `Property DecompositionID : SmallInt`

Visibility: public

Access: Read,Write

Description: Decomposition ID.

79.11.7 TUC_Prop.Category

Synopsis: Get the category.

Declaration: `Property Category : Byte`

Visibility: public

Access: Read,Write

Description: `CategoryData` provides access to the category part of `CategoryData` (encoded).

79.11.8 TUC_Prop.WhiteSpace

Synopsis: Is the character considered whitespace.

Declaration: `Property WhiteSpace : Boolean`

Visibility: public

Access: Read,Write

Description: `Whitespace` provides easy access to the `Whitespace` part of `CategoryData` (encoded).

79.11.9 TUC_Prop.HangulSyllable

Synopsis: Is the character a hangul syllable.

Declaration: `Property HangulSyllable : Boolean`

Visibility: public

Access: Read,Write

Description: `Whitespace` provides easy access to the `HangulSyllable` part of `CategoryData` (encoded).

79.11.10 TUC_Prop.UnifiedIdeograph

Declaration: `Property UnifiedIdeograph : Boolean`

Visibility: public

Access: Read

79.11.11 TUC_Prop.NumericValue

Synopsis: Numeric value.

Declaration: Property NumericValue : Double

Visibility: public

Access: Read

Description: NumericValue uses numericalindex to get the numerical value.

79.12 TUInt24Rec

```

TUInt24Rec = packed record
public
  c : Byte;
  b : Byte;
  a : Byte
;
  property byte0 : Byte;
  property byte1 : Byte;
  property byte2
    : Byte;
  TUInt24Rec.class operator implicit(a: TUInt24Rec) : Cardinal
;
  TUInt24Rec.class operator implicit(a: TUInt24Rec) : LongInt;
  TUInt24Rec.class operator implicit(a: TUInt24Rec) : Word;
  TUInt24Rec
    .class operator implicit(a: TUInt24Rec) : Byte;
  TUInt24Rec.class
    operator implicit(a: Cardinal) : TUInt24Rec;
  TUInt24Rec.class operator
    equal(a: TUInt24Rec; b: TUInt24Rec) : Boolean;
  TUInt24Rec.class
    operator equal(a: TUInt24Rec; b: Cardinal) : Boolean;
  TUInt24Rec
    .class operator equal(a: Cardinal; b: TUInt24Rec) : Boolean;
  TUInt24Rec
    .class operator equal(a: TUInt24Rec; b: LongInt) : Boolean;
  TUInt24Rec
    .class operator equal(a: LongInt; b: TUInt24Rec) : Boolean;
  TUInt24Rec
    .class operator equal(a: TUInt24Rec; b: Word) : Boolean;
  TUInt24Rec
    .class operator equal(a: Word; b: TUInt24Rec) : Boolean;
  TUInt24Rec
    .class operator equal(a: TUInt24Rec; b: Byte) : Boolean;
  TUInt24Rec
    .class operator equal(a: Byte; b: TUInt24Rec) : Boolean;
  TUInt24Rec
    .class operator notequal(a: TUInt24Rec; b: TUInt24Rec)
      : Boolean;
  TUInt24Rec.class operator

```

```

notequal(a: TUInt24Rec; b: Cardinal) : Boolean;
TUInt24Rec.class
operator notequal(a: Cardinal; b: TUInt24Rec) : Boolean;
TUInt24Rec
.class operator greaterthan(a: TUInt24Rec; b: TUInt24Rec)
: Boolean;
TUInt24Rec.class operator
greaterthan(a: TUInt24Rec; b: Cardinal)
: Boolean;
TUInt24Rec.class operator greaterthan
(a: Cardinal; b: TUInt24Rec)
: Boolean;
TUInt24Rec.class operator greaterthanorequal(a: TUInt24Rec
;
                                b: TUInt24Rec) :
Boolean;
TUInt24Rec.class operator greaterthanorequal(a: TUInt24Rec
; b: Cardinal)
                                : Boolean
;
TUInt24Rec.class operator greaterthanorequal(a: Cardinal; b: TUInt24Rec
)
                                : Boolean;
TUInt24Rec
.class operator lessthan(a: TUInt24Rec; b: TUInt24Rec)
: Boolean;
TUInt24Rec.class operator
lessthan(a: TUInt24Rec; b: Cardinal) : Boolean;
TUInt24Rec.class
operator lessthan(a: Cardinal; b: TUInt24Rec) : Boolean;
TUInt24Rec
.class operator lessthanorequal(a: TUInt24Rec; b: TUInt24Rec)
: Boolean;
TUInt24Rec.class
operator lessthanorequal(a: TUInt24Rec; b: Cardinal)
: Boolean;
TUInt24Rec.class operator
lessthanorequal(a: Cardinal; b: TUInt24Rec)
: Boolean;
end

```

Unicode data exists mostly of 24-bit data (3 bytes). This type is meant to deal efficiently with this data. It has members to split out the data in bytes, and functions to query the various properties stored in the data.

79.12.1 Method overview

Page	Method	Description
2184	<code>equal(Byte,TUInt24Rec):Boolean</code>	Check whether a TUInt24Rec value equals a byte value.
2183	<code>equal(Cardinal,TUInt24Rec):Boolean</code>	Check whether a TUInt24Rec value equals a cardinal value.
2183	<code>equal(LongInt,TUInt24Rec):Boolean</code>	Check whether a TUInt24Rec value equals a longint value.
2184	<code>equal(TUInt24Rec,Byte):Boolean</code>	Check whether a TUInt24Rec value equals a byte value.
2182	<code>equal(TUInt24Rec,Cardinal):Boolean</code>	Check if cardinal and TUInt24Rec are equal.
2183	<code>equal(TUInt24Rec,LongInt):Boolean</code>	Check whether a TUInt24Rec value equals a longint value.
2182	<code>equal(TUInt24Rec,TUInt24Rec):Boolean</code>	Determine equality of 2 TUInt24Rec records.
2183	<code>equal(TUInt24Rec,Word):Boolean</code>	Check whether a TUInt24Rec value equals a word value.
2183	<code>equal(Word,TUInt24Rec):Boolean</code>	Check whether a TUInt24Rec value equals a word value.
2185	<code>greaterthan(Cardinal,TUInt24Rec):Boolean</code>	Check whether a cardinal value is greater than a TUInt24Rec value.
2185	<code>greaterthan(TUInt24Rec,Cardinal):Boolean</code>	Check whether a TUInt24Rec value is greater than a cardinal value.
2185	<code>greaterthan(TUInt24Rec,TUInt24Rec):Boolean</code>	Check whether a TUInt24Rec value is greater than another TUInt24Rec value.
2186	<code>greaterthanorequal(Cardinal,TUInt24Rec):Boolean</code>	Check whether a cardinal value is greater than or equal to a TUInt24Rec value.
2185	<code>greaterthanorequal(TUInt24Rec,Cardinal):Boolean</code>	Check whether a TUInt24Rec value is greater than or equal to a cardinal value.
2185	<code>greaterthanorequal(TUInt24Rec,TUInt24Rec):Boolean</code>	Check whether a TUInt24Rec value is greater than or equal to a cardinal value.
2182	<code>implicit(Cardinal):TUInt24Rec</code>	Assign TUInt24Rec from Cardinal.
2182	<code>implicit(TUInt24Rec):Byte</code>	Assign TUInt24Rec to byte.
2181	<code>implicit(TUInt24Rec):Cardinal</code>	Assign TUInt24Rec to cardinal.
2181	<code>implicit(TUInt24Rec):LongInt</code>	Assign TUInt24Rec to longint.
2181	<code>implicit(TUInt24Rec):Word</code>	Assign TUInt24Rec to word.
2186	<code>lessthan(Cardinal,TUInt24Rec):Boolean</code>	Check whether a cardinal value is less than a TUInt24Rec value.
2186	<code>lessthan(TUInt24Rec,Cardinal):Boolean</code>	Check whether a

79.12.2 Property overview

Page	Properties	Access	Description
2187	byte0	rw	First byte.
2187	byte1	rw	Second byte.
2187	byte2	rw	Third byte.

79.12.3 TUInt24Rec.implicit(TUInt24Rec):Cardinal

Synopsis: Assign TUInt24Rec to cardinal.

Declaration: `TUInt24Rec.class operator implicit(a: TUInt24Rec) : Cardinal`

Visibility: public

Description: Assign to cardinal, byte0 to MSB and so on.

See also: `TUInt24Rec.implicit(TUInt24Rec):LongInt` ([2181](#)), `TUInt24Rec.implicit(TUInt24Rec):Word` ([2181](#)), `TUInt24Rec.implicit(TUInt24Rec):Byte` ([2182](#)), `TUInt24Rec.implicit(Cardinal):TUInt24Rec` ([2182](#)), `TUInt24Rec.implicit(Longint):TUInt24Rec` ([2178](#)), `TUInt24Rec.implicit(Word):TUInt24Rec` ([2178](#)), `TUInt24Rec.implicit(Byte):TUInt24Rec` ([2178](#))

79.12.4 TUInt24Rec.implicit(TUInt24Rec):LongInt

Synopsis: Assign TUInt24Rec to longint.

Declaration: `TUInt24Rec.class operator implicit(a: TUInt24Rec) : LongInt`

Visibility: public

Description: Assign to cardinal, byte0 to MSB and so on.

See also: `TUInt24Rec.implicit(TUInt24Rec):Cardinal` ([2181](#)), `TUInt24Rec.implicit(TUInt24Rec):Word` ([2181](#)), `TUInt24Rec.implicit(TUInt24Rec):Byte` ([2182](#))

79.12.5 TUInt24Rec.implicit(TUInt24Rec):Word

Synopsis: Assign TUInt24Rec to word.

Declaration: `TUInt24Rec.class operator implicit(a: TUInt24Rec) : Word`

Visibility: public

Description: Assign to word, byte0 to MSB, byte1 to MSB.

Errors: If the value is too big (>\$FFFF) to be assigned, an overflow error occurs.

See also: `TUInt24Rec.implicit(TUInt24Rec):Cardinal` ([2181](#)), `TUInt24Rec.implicit(TUInt24Rec):Longint` ([2181](#)), `TUInt24Rec.implicit(TUInt24Rec):Byte` ([2182](#)), `TUInt24Rec.implicit(Cardinal):TUInt24Rec` ([2182](#)), `TUInt24Rec.implicit(Longint):TUInt24Rec` ([2178](#)), `TUInt24Rec.implicit(Word):TUInt24Rec` ([2178](#)), `TUInt24Rec.implicit(Byte):TUInt24Rec` ([2178](#))

79.12.6 TUInt24Rec.implicit(TUInt24Rec):Byte

Synopsis: Assign TUInt24Rec to byte.

Declaration: `TUInt24Rec.class operator implicit(a: TUInt24Rec) : Byte`

Visibility: public

Description: Assign to byte, byte0 is assigned.

Errors: If the value is too big (>\$FF) to be assigned, an overflow error occurs.

See also: `TUInt24Rec.implicit(TUInt24Rec):Cardinal` (2181), `TUInt24Rec.implicit(TUInt24Rec):Longint` (2181), `TUInt24Rec.implicit(TUInt24Rec):Word` (2181), `TUInt24Rec.implicit(Cardinal):TUInt24Rec` (2182), `TUInt24Rec.implicit(Longint):TUInt24Rec` (2178), `TUInt24Rec.implicit(Word):TUInt24Rec` (2178), `TUInt24Rec.implicit(Byte):TUInt24Rec` (2178)

79.12.7 TUInt24Rec.implicit(Cardinal):TUInt24Rec

Synopsis: Assign TUInt24Rec from Cardinal.

Declaration: `TUInt24Rec.class operator implicit(a: Cardinal) : TUInt24Rec`

Visibility: public

Description: Assign from cardinal, byte0 to MSB.

Errors: If the value is too big (>\$FFFFFF) to be assigned, an overflow error occurs.

See also: `TUInt24Rec.implicit(TUInt24Rec):Longint` (2181), `TUInt24Rec.implicit(TUInt24Rec):Word` (2181), `TUInt24Rec.implicit(TUInt24Rec):Byte` (2182), `TUInt24Rec.implicit(Longint):TUInt24Rec` (2178), `TUInt24Rec.implicit(Word):TUInt24Rec` (2178), `TUInt24Rec.implicit(Byte):TUInt24Rec` (2178)

79.12.8 TUInt24Rec.equal(TUInt24Rec,TUInt24Rec):Boolean

Synopsis: Determine equality of 2 TUInt24Rec records.

Declaration: `TUInt24Rec.class operator equal(a: TUInt24Rec; b: TUInt24Rec) : Boolean`

Visibility: public

Description: The 2 records are considered equal if the 3 bytes are equal.

79.12.9 TUInt24Rec.equal(TUInt24Rec,Cardinal):Boolean

Synopsis: Check if cardinal and TUInt24Rec are equal.

Declaration: `TUInt24Rec.class operator equal(a: TUInt24Rec; b: Cardinal) : Boolean`

Visibility: public

Description: The cardinal b is considered equal to a if the fourth byte (LSB) is zero, and the first three bytes equal byte0, byte1, and byte2.

79.12.10 `TUInt24Rec.equal(Cardinal,TUInt24Rec):Boolean`

Synopsis: Check whether a `TUInt24Rec` value equals a cardinal value.

Declaration: `TUInt24Rec.class operator equal(a: Cardinal; b: TUInt24Rec) : Boolean`

Visibility: public

Description: The cardinal `a` is considered equal to `b` if the fourth byte (LSB) is zero, and the first three bytes equal `byte0`, `byte1`, and `byte2`.

79.12.11 `TUInt24Rec.equal(TUInt24Rec,LongInt):Boolean`

Synopsis: Check whether a `TUInt24Rec` value equals a longint value.

Declaration: `TUInt24Rec.class operator equal(a: TUInt24Rec; b: LongInt) : Boolean`

Visibility: public

Description: The longint `b` is considered equal to `a` if the fourth byte (LSB) is zero, and the first three bytes equal `byte0`, `byte1`, and `byte2`.

79.12.12 `TUInt24Rec.equal(LongInt,TUInt24Rec):Boolean`

Synopsis: Check whether a `TUInt24Rec` value equals a longint value.

Declaration: `TUInt24Rec.class operator equal(a: LongInt; b: TUInt24Rec) : Boolean`

Visibility: public

Description: The longint `a` is considered equal to `b` if the fourth byte (LSB) is zero, and the first three bytes equal `byte0`, `byte1`, and `byte2`.

79.12.13 `TUInt24Rec.equal(TUInt24Rec,Word):Boolean`

Synopsis: Check whether a `TUInt24Rec` value equals a word value.

Declaration: `TUInt24Rec.class operator equal(a: TUInt24Rec; b: Word) : Boolean`

Visibility: public

Description: The word `b` is considered equal to `a` if its 2 bytes equal `byte0`, `byte1` and `byte2` is zero.

79.12.14 `TUInt24Rec.equal(Word,TUInt24Rec):Boolean`

Synopsis: Check whether a `TUInt24Rec` value equals a word value.

Declaration: `TUInt24Rec.class operator equal(a: Word; b: TUInt24Rec) : Boolean`

Visibility: public

Description: The word `a` is considered equal to `b` if its 2 bytes equal `byte0`, `byte1` and `byte2` is zero.

79.12.15 TUInt24Rec.equal(TUInt24Rec,Byte):Boolean

Synopsis: Check whether a TUInt24Rec value equals a byte value.

Declaration: `TUInt24Rec.class operator equal(a: TUInt24Rec; b: Byte) : Boolean`

Visibility: public

Description: The byte `b` is considered equal to `a` if it equals `byte0` and `byte1` and `byte1` are zero.

79.12.16 TUInt24Rec.equal(Byte,TUInt24Rec):Boolean

Synopsis: Check whether a TUInt24Rec value equals a byte value.

Declaration: `TUInt24Rec.class operator equal(a: Byte; b: TUInt24Rec) : Boolean`

Visibility: public

Description: The byte `b` is considered equal to `a` if it equals `byte0` and `byte1` and `byte1` are zero.

79.12.17 TUInt24Rec.notequal(TUInt24Rec,TUInt24Rec):Boolean

Synopsis: Check whether 2 TUInt24Rec values differ.

Declaration: `TUInt24Rec.class operator notequal(a: TUInt24Rec; b: TUInt24Rec)
: Boolean`

Visibility: public

Description: The comparison is done by comparing `byte0`, `byte1` and `byte2`.

79.12.18 TUInt24Rec.notequal(TUInt24Rec,Cardinal):Boolean

Synopsis: Check whether a TUInt24Rec value differs from a cardinal value.

Declaration: `TUInt24Rec.class operator notequal(a: TUInt24Rec; b: Cardinal) : Boolean`

Visibility: public

Description: The cardinal `a` is considered not equal to `b` if the fourth byte (LSB) is nonzero, or one of the first three bytes differ from `byte0`, `byte1`, and `byte2`.

79.12.19 TUInt24Rec.notequal(Cardinal,TUInt24Rec):Boolean

Synopsis: Check whether a TUInt24Rec value differs from a cardinal value.

Declaration: `TUInt24Rec.class operator notequal(a: Cardinal; b: TUInt24Rec) : Boolean`

Visibility: public

Description: The cardinal `a` is considered not equal to `b` if the fourth byte (LSB) is nonzero, or one of the first three bytes differ from `byte0`, `byte1`, and `byte2`.

79.12.20 **TUInt24Rec.greaterthan(TUInt24Rec,TUInt24Rec):Boolean**

Synopsis: Check whether a TUInt24Rec value is greater than another TUInt24Rec value.

Declaration: `TUInt24Rec.class operator greaterthan(a: TUInt24Rec; b: TUInt24Rec)
: Boolean`

Visibility: public

Description: The comparison is done by comparing 3 bytes `byte0`, `byte1` and `byte2`

79.12.21 **TUInt24Rec.greaterthan(TUInt24Rec,Cardinal):Boolean**

Synopsis: Check whether a TUInt24Rec value is greater than a cardinal value.

Declaration: `TUInt24Rec.class operator greaterthan(a: TUInt24Rec; b: Cardinal)
: Boolean`

Visibility: public

Description: The comparison is done by comparing 3 first bytes of the cardinal value with bytes `byte0`, `byte1` and `byte2`.

79.12.22 **TUInt24Rec.greaterthan(Cardinal,TUInt24Rec):Boolean**

Synopsis: Check whether a cardinal value is greater than a TUInt24Rec value.

Declaration: `TUInt24Rec.class operator greaterthan(a: Cardinal; b: TUInt24Rec)
: Boolean`

Visibility: public

Description: The comparison is done by comparing 3 first bytes of the cardinal value with bytes `byte0`, `byte1` and `byte2`.

79.12.23 **TUInt24Rec.greaterthanorequal(TUInt24Rec,TUInt24Rec):Boolean**

Synopsis: Check whether a TUInt24Rec value is greater than or equal to a cardinal value.

Declaration: `TUInt24Rec.class operator greaterthanorequal(a: TUInt24Rec;
b: TUInt24Rec) : Boolean`

Visibility: public

Description: The comparison is done by comparing 3 first bytes of the cardinal value with bytes `byte0`, `byte1` and `byte2`.

79.12.24 **TUInt24Rec.greaterthanorequal(TUInt24Rec,Cardinal):Boolean**

Synopsis: Check whether a TUInt24Rec value is greater than or equal to a cardinal value.

Declaration: `TUInt24Rec.class operator greaterthanorequal(a: TUInt24Rec; b: Cardinal)
: Boolean`

Visibility: public

Description: The comparison is done by comparing 3 first bytes of the cardinal value with bytes `byte0`, `byte1` and `byte2`.

79.12.25 TUInt24Rec.greaterthanorequal(Cardinal,TUInt24Rec):Boolean

Synopsis: Check whether a cardinal value is greater than or equal to a TUInt24Rec value.

Declaration: `TUInt24Rec.class operator greaterthanorequal(a: Cardinal; b: TUInt24Rec)
: Boolean`

Visibility: public

Description: The comparison is done by comparing 3 first bytes of the cardinal value with bytes `byte0`, `byte1` and `byte2`.

79.12.26 TUInt24Rec.lesssthan(TUInt24Rec,TUInt24Rec):Boolean

Synopsis: Check whether a TUInt24Rec value is less than another TUInt24Rec value.

Declaration: `TUInt24Rec.class operator lesssthan(a: TUInt24Rec; b: TUInt24Rec)
: Boolean`

Visibility: public

Description: The comparison is done by comparing the bytes `byte0`, `byte1` and `byte2`.

79.12.27 TUInt24Rec.lesssthan(TUInt24Rec,Cardinal):Boolean

Synopsis: Check whether a TUInt24Rec value is less than a cardinal value.

Declaration: `TUInt24Rec.class operator lesssthan(a: TUInt24Rec; b: Cardinal) : Boolean`

Visibility: public

Description: The comparison is done by comparing 3 first bytes of the cardinal value with bytes `byte0`, `byte1` and `byte2`.

79.12.28 TUInt24Rec.lesssthan(Cardinal,TUInt24Rec):Boolean

Synopsis: Check whether a cardinal value is less than a TUInt24Rec value.

Declaration: `TUInt24Rec.class operator lesssthan(a: Cardinal; b: TUInt24Rec) : Boolean`

Visibility: public

Description: Check whether a cardinal value is less than a TUInt24Rec value.

79.12.29 TUInt24Rec.lesssthanorequal(TUInt24Rec,TUInt24Rec):Boolean

Synopsis: Check whether a TUInt24Rec value is less than or equal to another TUInt24Rec value.

Declaration: `TUInt24Rec.class operator lesssthanorequal(a: TUInt24Rec; b: TUInt24Rec)
: Boolean`

Visibility: public

Description: The comparison is done by comparing the bytes `byte0`, `byte1` and `byte2`.

79.12.30 **TUInt24Rec.lessthanequal(TUInt24Rec,Cardinal):Boolean**

Synopsis: Check whether a TUInt24Rec value is less than or equal to a cardinal value.

Declaration: `TUInt24Rec.class operator lessthanequal(a: TUInt24Rec; b: Cardinal)
: Boolean`

Visibility: public

Description: The comparison is done by comparing 3 first bytes of the cardinal value with bytes `byte0`, `byte1` and `byte2`.

79.12.31 **TUInt24Rec.lessthanequal(Cardinal,TUInt24Rec):Boolean**

Synopsis: Check whether a cardinal value is less than or equal to a TUInt24Rec value.

Declaration: `TUInt24Rec.class operator lessthanequal(a: Cardinal; b: TUInt24Rec)
: Boolean`

Visibility: public

Description: The comparison is done by comparing 3 first bytes of the cardinal value with bytes `byte0`, `byte1` and `byte2`.

79.12.32 **TUInt24Rec.byte0**

Synopsis: First byte.

Declaration: `Property byte0 : Byte`

Visibility: public

Access: Read,Write

79.12.33 **TUInt24Rec.byte1**

Synopsis: Second byte.

Declaration: `Property byte1 : Byte`

Visibility: public

Access: Read,Write

79.12.34 **TUInt24Rec.byte2**

Synopsis: Third byte.

Declaration: `Property byte2 : Byte`

Visibility: public

Access: Read,Write

Chapter 80

Reference for unit 'unicodeducet'

80.1 Used units

Table 80.1: Used units by unit 'unicodeducet'

Name	Page
System	1335

80.2 Overview

The `unicodeducet` unit registers the root Unicode collation (DUCET). This collation is needed by all other collations, so any collation unit will include this file.

This unit does not contain any routines. It simply registers the collation in the initialization section of the unit, so including the unit in the `uses` clause of the program is sufficient.

Chapter 81

Reference for unit 'Unix'

81.1 Used units

Table 81.1: Used units by unit 'Unix'

Name	Page
BaseUnix	155
System	1335
unixtype	2229

81.2 Constants, types and variables

81.2.1 Constants

`ARG_MAX = UnixType.ARG_MAX`

Maximum number of arguments to a program.

`fs_ext = $137d`

File system type (TStatFS ([2204](#))): (ext) Extended.

`fs_ext2 = $ef53`

File system type (TStatFS ([2204](#))): (ext2) Second extended.

`fs_iso = $9660`

File system type (TStatFS ([2204](#))): ISO 9660.

`fs_minix = $137f`

File system type (TStatFS ([2204](#))): Minix.

`fs_minix_30 = $138f`

File system type (TStatFS (2204)): Minix 3.0.

`fs_minix_V2 = $2468`

File system type (TStatFS (2204)): Minix V2.

`fs_msdos = $4d44`

File system type (TStatFS (2204)): MSDOS (FAT).

`fs_nfs = $6969`

File system type (TStatFS (2204)): NFS.

`fs_old_ext2 = $ef51`

File system type (TStatFS (2204)): (ext2) Old second extended.

`fs_proc = $9fa0`

File system type (TStatFS (2204)): PROC fs.

`fs_xia = $012FD16D`

File system type (TStatFS (2204)): XIA.

`IOctl_TCGETS = $5401`

IOCTL call number: get Terminal Control settings.

`LOCK_EX = 2`

FpFLock (2213) Exclusive lock.

`LOCK_NB = 4`

FpFLock (2213) Non-blocking operation.

`LOCK_SH = 1`

FpFLock (2213) Shared lock.

`LOCK_UN = 8`

FpFLock (2213) unlock.

`MAP_FAILED = baseunix.MAP_FAILED`

Error return value for mmap: mmap operation failed.

`MAP_FIXED = baseunix.MAP_FIXED`

#rtl.baseunix.FpMMap (216) map type: Interpret addr exactly.

MAP_PRIVATE = baseunix.MAP_PRIVATE

#rtl.baseunix.FpMMap (216) map type: Changes are private.

MAP_SHARED = baseunix.MAP_SHARED

#rtl.baseunix.FpMMap (216) map type: Share changes.

MAP_TYPE = baseunix.MAP_TYPE

#rtl.baseunix.FpMMap (216) map type: Bitmask for type of mapping.

NAME_MAX = UnixType.NAME_MAX

Maximum filename length.

Open_Accmode = 3

Bitmask to determine access mode in open flags.

Open_Append = 2 shl 9

File open mode: Append to file.

Open_Creat = 1 shl 6

File open mode: Create if file does not yet exist.

Open_Direct = 4 shl 12

File open mode: Minimize caching effects.

Open_Directory = 2 shl 15

File open mode: File must be directory.

Open_Excl = 2 shl 6

File open mode: Open exclusively.

Open_LargeFile = 1 shl 15

File open mode: Open for 64-bit I/O.

Open_NDelay = Open_NonBlock

File open mode: Alias for Open_NonBlock (2192).

Open_NoCtty = 4 shl 6

File open mode: No TTY control.

`Open_NoFollow = 4 shl 15`

File open mode: Fail if file is symbolic link.

`Open_NonBlock = 4 shl 9`

File open mode: Open in non-blocking mode.

`Open_RdOnly = 0`

File open mode: Read only.

`Open_RdWr = 2`

File open mode: Read/Write.

`Open_Sync = 1 shl 12`

File open mode: Write to disc at once.

`Open_Trunc = 1 shl 9`

File open mode: Truncate file to length 0.

`Open_WrOnly = 1`

File open mode: Write only.

`PATH_MAX = UnixType.PATH_MAX`

Maximum pathname length.

`PRIO_PGRP = UnixType.PRIO_PGRP`

`#rtl.baseunix.fpGetPriority (210)` option: Get process group priority.

`PRIO_PROCESS = UnixType.PRIO_PROCESS`

`#rtl.baseunix.fpGetPriority (210)` option: Get process priority.

`PRIO_USER = UnixType.PRIO_USER`

`#rtl.baseunix.fpGetPriority (210)` option: Get user priority.

`PROT_EXEC = baseunix.PROT_EXEC`

`#rtl.baseunix.FpMMap (216)` memory access: page can be executed.

`PROT_NONE = baseunix.PROT_NONE`

#rtl.baseunix.FpMMap (216) memory access: page can not be accessed.

PROT_READ = baseunix.PROT_READ

#rtl.baseunix.FpMMap (216) memory access: page can be read.

PROT_WRITE = baseunix.PROT_WRITE

#rtl.baseunix.FpMMap (216) memory access: page can be written.

P_IN = 1

Input file descriptor of pipe pair.

P_OUT = 2

Output file descriptor of pipe pair.

SIG_MAXSIG = UnixType.SIG_MAXSIG

Maximum system signal number.

STAT_IFBLK = \$6000

File (#rtl.baseunix.stat (249) record) mode: Block device.

STAT_IFCHR = \$2000

File (#rtl.baseunix.stat (249) record) mode: Character device.

STAT_IFDIR = \$4000

File (#rtl.baseunix.stat (249) record) mode: Directory.

STAT_IFIFO = \$1000

File (#rtl.baseunix.stat (249) record) mode: FIFO.

STAT_IFLNK = \$a000

File (#rtl.baseunix.stat (249) record) mode: Link.

STAT_IFMT = \$f000

File (#rtl.baseunix.stat (249) record) mode: File type bit mask.

STAT_IFREG = \$8000

File (#rtl.baseunix.stat (249) record) mode: Regular file.

STAT_IFSOCK = \$c000

File (#rtl.baseunix.stat (249) record) mode: Socket.

STAT_IRGRP = STAT_IROTH shl 3

File (#rtl.baseunix.stat (249) record) mode: Group read permission.

STAT_IROTH = \$4

File (#rtl.baseunix.stat (249) record) mode: Other read permission.

STAT_IRUSR = STAT_IROTH shl 6

File (#rtl.baseunix.stat (249) record) mode: Owner read permission.

STAT_IRWXG = STAT_IRWXO shl 3

File (#rtl.baseunix.stat (249) record) mode: Group permission bits mask.

STAT_IRWXO = \$7

File (#rtl.baseunix.stat (249) record) mode: Other permission bits mask.

STAT_IRWXU = STAT_IRWXO shl 6

File (#rtl.baseunix.stat (249) record) mode: Owner permission bits mask.

STAT_ISGID = \$0400

File (#rtl.baseunix.stat (249) record) mode: GID bit set.

STAT_ISUID = \$0800

File (#rtl.baseunix.stat (249) record) mode: UID bit set.

STAT_ISVTX = \$0200

File (#rtl.baseunix.stat (249) record) mode: Sticky bit set.

STAT_IWGRP = STAT_IWOTH shl 3

File (#rtl.baseunix.stat (249) record) mode: Group write permission.

STAT_IWOTH = \$2

File (#rtl.baseunix.stat (249) record) mode: Other write permission.

STAT_IWUSR = STAT_IWOTH shl 6

File (#rtl.baseunix.stat (249) record) mode: Owner write permission.

STAT_IXGRP = STAT_IXOTH shl 3

File (`#rtl.baseunix.stat` (249) record) mode: Others execute permission.

```
STAT_IXOTH = $1
```

File (`#rtl.baseunix.stat` (249) record) mode: Others execute permission.

```
STAT_IXUSR = STAT_IXOTH shl 6
```

File (`#rtl.baseunix.stat` (249) record) mode: Others execute permission.

```
SYS_NMLN = UnixType.SYS_NMLN
```

Max system name length.

```
Wait_Any = - 1
```

`#rtl.baseunix.fpWaitPID` (245): Wait on any process.

```
Wait_Clone = $800000000
```

`#rtl.baseunix.fpWaitPID` (245): Wait on clone processes only.

```
Wait_MyPGRP = 0
```

`#rtl.baseunix.fpWaitPID` (245): Wait processes from current process group.

```
Wait_NoHang = 1
```

`#rtl.baseunix.fpWaitPID` (245): Do not wait.

```
Wait_UnTraced = 2
```

`#rtl.baseunix.fpWaitPID` (245): Also report stopped but untraced processes.

81.2.2 Types

```
cbool = UnixType.cbool
```

Boolean type.

```
cchar = UnixType.cchar
```

Alias for `#rtl.UnixType.cchar` (2231).

```
cdouble = UnixType.cdouble
```

Double precision real format.

```
cfloat = UnixType.cfloat
```

Floating-point real format.

```
cint = UnixType.cint
```

C type: integer (natural size).

```
cint16 = UnixType.cint16
```

C type: 16 bits sized, signed integer.

```
cint32 = UnixType.cint32
```

C type: 32 bits sized, signed integer.

```
cint64 = UnixType.cint64
```

C type: 64 bits sized, signed integer.

```
cint8 = UnixType.cint8
```

C type: 8 bits sized, signed integer.

```
clock_t = UnixType.clock_t
```

Clock ticks type.

```
clong = UnixType.clong
```

C type: long signed integer (double sized).

```
clonglong = UnixType.clonglong
```

C type: 64-bit (double long) signed integer.

```
coff_t = UnixType.TOff
```

character offset type.

```
cschar = UnixType.cschar
```

Signed character type.

```
cshort = UnixType.cshort
```

C type: short signed integer (half sized).

```
csigned = UnixType.csigned
```

`csigned` is an alias for `cint` ([2196](#)).

```
csint = UnixType.csint
```

Signed integer.

```
csize_t = UnixType.size_t
```

Character size type.

```
cslong = UnixType.cslong
```

The size is CPU dependent.

```
cslonglong = UnixType.cslonglong
```

cslonglong is an alias for clonglong ([2196](#)).

```
csshort = UnixType.csshort
```

Short signed integer type.

```
cuchar = UnixType.cuchar
```

Alias for #rtl.UnixType.cuchar ([2232](#)).

```
cuint = UnixType.cuint
```

C type: unsigned integer (natural size).

```
cuint16 = UnixType.cuint16
```

C type: 16 bits sized, unsigned integer.

```
cuint32 = UnixType.cuint32
```

C type: 32 bits sized, unsigned integer.

```
cuint64 = UnixType.cuint64
```

C type: 64 bits sized, unsigned integer.

```
cuint8 = UnixType.cuint8
```

C type: 8 bits sized, unsigned integer.

```
culong = UnixType.culong
```

C type: long unsigned integer (double sized).

```
culonglong = UnixType.culonglong
```

C type: 64-bit (double long) unsigned integer.

```
cunsigned = UnixType.cunsigned
```

Alias for `#rtl.unixtype.cunsigned` (2233).

```
cushort = UnixType.cushort
```

C type: short unsigned integer (half sized).

```
dev_t = UnixType.dev_t
```

Device descriptor type.

```
gid_t = UnixType.gid_t
```

Group ID type.

```
ino_t = UnixType.ino_t
```

Inode type.

```
mode_t = UnixType.mode_t
```

Inode mode type.

```
nlink_t = UnixType.nlink_t
```

Number of links type.

```
off_t = UnixType.off_t
```

Offset type.

```
pcbool = UnixType.pcbool
```

Pointer to boolean type `cbool` (2195)

```
pcchar = UnixType.pcchar
```

Alias for `#rtl.UnixType.pcchar` (2234).

```
pcdouble = UnixType.pcdouble
```

Pointer to `cdouble` (180) type.

```
pcfloat = UnixType.pcfloating
```

Pointer to `cfloat` (180) type.

```
pcint = UnixType.pcint
```

Pointer to `cInt` (2196) type.

```
pcint16 = UnixType.pcint16
```

Pointer to 16-bit signed integer type.

```
pcint32 = UnixType.pcint32
```

Pointer to signed 32-bit integer type.

```
pcint64 = UnixType.pcint64
```

Pointer to signed 64-bit integer type.

```
pcint8 = UnixType.pcint8
```

Pointer to 8-bits signed integer type.

```
pClock = UnixType.pClock
```

Pointer to TClock ([2202](#)) type.

```
pclong = UnixType.pclong
```

Pointer to cLong ([2196](#)) type.

```
pclonglong = UnixType.pclonglong
```

Pointer to longlong type.

```
pcschar = UnixType.pcschar
```

Pointer to character type cschar ([2196](#)).

```
pcshort = UnixType.pcsshort
```

Pointer to cShort ([2196](#)) type.

```
pcsigned = UnixType.pcsigned
```

Pointer to signed integer type csigned ([2196](#)).

```
pcsint = UnixType.pcsint
```

Pointer to signed integer type csint ([2197](#))

```
pctype_t = UnixType.pctype_t
```

Pointer to character size type pctype_t.

```
pclong = UnixType.pclong
```

Pointer to the signed long clong ([2197](#))

```
pclonglong = UnixType.pclonglong
```


Pointer to Signed longlong type cslonglong ([2197](#))

```
pcsshort = UnixType.pcsshort
```

Pointer to short signed integer type csshort ([2197](#))

```
pcuchar = UnixType.pcuchar
```

Alias for #rtl.UnixType.pcuchar ([2235](#)).

```
pcuint = UnixType.pcuint
```

Pointer to cUInt ([2197](#)) type.

```
pcuint16 = UnixType.pcuint16
```

Pointer to 16-bit unsigned integer type.

```
pcuint32 = UnixType.pcuint32
```

Pointer to unsigned 32-bit integer type.

```
pcuint64 = UnixType.pcuint64
```

Pointer to unsigned 64-bit integer type.

```
pcuint8 = UnixType.pcuint8
```

Pointer to 8-bits unsigned integer type.

```
pculong = UnixType.pculong
```

Pointer to cuLong ([2197](#)) type.

```
pculonglong = UnixType.pculonglong
```

Unsigned longlong type.

```
pcunsigned = UnixType.pcunsigned
```

Alias for #rtl.unixtype.pcunsigned ([2236](#)).

```
pcushort = UnixType.pcushort
```

Pointer to cuShort ([2198](#)) type.

```
pDev = UnixType.pDev
```

Pointer to TDev ([2202](#)) type.

```
pGid = UnixType.pGid
```

Pointer to TGid (2203) type.

```
pid_t = UnixType.pid_t
```

Process ID type.

```
pIno = UnixType.pIno
```

Pointer to TIno (2203) type.

```
pMode = UnixType.pMode
```

Pointer to TMode (2203) type.

```
pnLink = UnixType.pnLink
```

Pointer to TnLink (2203) type.

```
pOff = UnixType.pOff
```

Pointer to TOff (2203) type.

```
pPid = UnixType.pPid
```

Pointer to TPid (2203) type.

```
pSize = UnixType.pSize
```

Pointer to TSize (2204) type.

```
pSize_t = UnixType.pSize_t
```

Pointer to type Size_t.

```
pSocklen = UnixType.pSocklen
```

Pointer to TSockLen (2204) type.

```
psSize = UnixType.psSize
```

Pointer to TsSize (2204) type.

```
pstatfs = UnixType.PStatFs
```

Pointer to statfs type.

```
pthread_cond_t = UnixType.pthread_cond_t
```

Thread conditional variable type.

```
pthread_mutex_t = UnixType.pthread_mutex_t
```

Thread mutex type.

```
pthread_t = UnixType.pthread_t
```

POSIX thread type.

```
pTime = UnixType.pTime
```

Pointer to TTime (2204) type.

```
ptimespec = UnixType.ptimespec
```

Pointer to timespec (2203) type.

```
ptimeval = UnixType.ptimeval
```

Pointer to timeval (2203) type.

```
ptime_t = UnixType.ptime_t
```

Pointer to time_t (2203) type.

```
pUId = UnixType.pUId
```

Pointer to TUid (2204) type.

```
size_t = UnixType.size_t
```

Size specification type.

```
socklen_t = UnixType.socklen_t
```

Socket address length type.

```
ssize_t = UnixType.ssize_t
```

Small size type.

```
TClock = UnixType.TClock
```

Alias for clock_t (2196) type.

```
TDev = UnixType.TDev
```

Alias for dev_t (2198) type.

```
TFSearchOption = (NoCurrentDirectory, CurrentDirectoryFirst,  
    CurrentDirectoryLast)
```

Table 81.2: Enumeration values for type TFSearchOption

Value	Explanation
CurrentDirectoryFirst	Search the current directory first, before all directories in the search path.
CurrentDirectoryLast	Search the current directory last, after all directories in the search path.
NoCurrentDirectory	Do not search the current directory unless it is specified in the search path.

Describes the search strategy used by FSearch ([2215](#)).

`TGid = UnixType.TGid`

Alias for `gid_t` ([2198](#)) type.

`timespec = UnixType.timespec`

Short time specification type.

`timeval = UnixType.timeval`

Time specification type.

`time_t = UnixType.time_t`

Time span type.

`TIno = UnixType.TIno`

Alias for `ino_t` ([2198](#)) type.

`TIOctlRequest = UnixType.TIOctlRequest`

Alias for the `TIOctlRequest` ([2239](#)) type in `unixtypes`.

`TMode = UnixType.TMode`

Alias for `mode_t` ([2198](#)) type.

`TnLink = UnixType.TnLink`

Alias for `nlink_t` ([2198](#)) type.

`TOff = UnixType.TOff`

Alias for `off_t` ([2198](#)) type.

`TPid = UnixType.TPid`

Alias for `pid_t` ([2201](#)) type.

`TSize = UnixType.TSize`

Alias for `size_t` (2202) type.

`TSocklen = UnixType.TSocklen`

Alias for `socklen_t` (2202) type.

`TsSize = UnixType.TsSize`

Alias for `ssize_t` (2202) type.

`tstatfs = UnixType.TStatFs`

`StatFS` returns in `Info` information about the file system on which the file `Path` resides. `Info` is of type `TStatFS` (2243).

The function returns zero if the call was successful, a nonzero value is returned if the call failed.

`TTime = UnixType.TTime`

Alias for `TTime` (2204) type.

`Ttimespec = UnixType.Ttimespec`

Alias for `TimeSpec` (2203) type.

`TTimeVal = UnixType.TTimeVal`

Alias for `timeval` (2203) type.

`TUid = UnixType.TUid`

Alias for `uid_t` (2204) type.

`uid_t = UnixType.uid_t`

User ID type.

81.3 Procedures and functions

81.3.1 AssignPipe

Synopsis: Create a set of pipe file handlers.

Declaration: `function AssignPipe(var pipe_in: cint; var pipe_out: cint) : cint`
`function AssignPipe(var pipe_in: text; var pipe_out: text) : cint`
`function AssignPipe(var pipe_in: File; var pipe_out: File) : cint`

Visibility: default

Description: `AssignPipe` creates a pipe, i.e. two file objects, one for input, one for output. What is written to `Pipe_out`, can be read from `Pipe_in`.

This call is overloaded. The in and out pipe can take three forms: an typed or untyped file, a text file or a file descriptor.

If a text file is passed then reading and writing from/to the pipe can be done through the usual `Readln(Pipe_in, ...)` and `Writeln(Pipe_out, ...)` procedures.

The function returns `True` if everything went successfully, `False` otherwise.

Errors: In case the function fails and returns `False`, extended error information is returned by the `FpGetErrno` (207) function:

sys_emfile Too many file descriptors for this process.

sys_enfile The system file table is full.

See also: `POpen` (2219), `#rtl.baseunix.FpMkFifo` (216)

Listing: `./examples/unixex/ex36.pp`

Program Example36;

{ Program to demonstrate the AssignPipe function. }

Uses BaseUnix, Unix;

Var pipi, pipo : Text;
s : String;

```
begin
  Writeln ('Assigning Pipes. ');
  If assignpipe(pipi, pipo) <> 0 then
    Writeln ('Error assigning pipes !', fpgeterrno);
  Writeln ('Writing to pipe, and flushing. ');
  Writeln (pipo, 'This is a textstring'); close(pipo);
  Writeln ('Reading from pipe. ');
  While not eof(pipi) do
    begin
      Readln (pipi, s);
      Writeln ('Read from pipe : ', s);
    end;
  close (pipi);
  writeln ('Closed pipes. ');
  writeln
end.
```

81.3.2 AssignStream

Synopsis: Assign stream for in and output to a program.

Declaration:

```
function AssignStream(var StreamIn: text; var Streamout: text;
                     const Prog: ansiString;
                     const args: Array of ansistring) : cint
function AssignStream(var StreamIn: text; var Streamout: text;
                     var streamerr: text; const Prog: ansiString;
                     const args: Array of ansistring) : cint
```

Visibility: default

Description: `AssignStream` creates a 2 or 3 pipes, i.e. two (or three) file objects, one for input, one for output, (and one for standard error) the other ends of these pipes are connected to standard input and output (and standard error) of `Prog`. `Prog` is the path of a program (including path). The options for the program can be specified in `Args`.

What is written to `StreamOut`, will go to the standard input of `Prog`. Whatever is written by `Prog` to its standard output can be read from `StreamIn`. Whatever is written by `Prog` to its standard error read from `StreamErr`, if present.

Reading and writing happens through the usual `Readln(StreamIn, ...)` and `Writeln(StreamOut, ...)` procedures.

Remark You should *not* use `Reset` or `Rewrite` on a file opened with `POpen`. This will close the file before re-opening it again, thereby closing the connection with the program.

The function returns the process ID of the spawned process, or -1 in case of error.

Errors: Extended error information is returned by the `FpGetErrno` (207) function.

sys_emfile Too many file descriptors for this process.

sys_enfile The system file table is full.

Other errors include the ones by the fork and exec programs

See also: `AssignPipe` (2204), `POpen` (2219)

Listing: `./examples/unixex/ex38.pp`

Program Example38;

{ Program to demonstrate the AssignStream function. }

Uses BaseUnix, Unix;

Var Si, So : Text;

 S : String;

 i : longint;

begin

if not (paramstr(1)='-son') **then**

begin

Writeln ('Calling son');

 Assignstream (Si, So, paramstr(0), ['-son']);

if fpgeterrno <> 0 **then**

begin

writeln ('AssignStream failed !');

 halt(1);

end;

Writeln ('Speaking to son');

For i:=1 **to** 10 **do**

begin

writeln (so, 'Hello son !');

if iresult <> 0 **then writeln** ('Can"t speak to son...');

end;

For i:=1 **to** 3 **do writeln** (so, 'Hello chap !');

 close (so);

while not eof(si) **do**

begin

```

    readln (si,s);
    writeln ('Father: Son said : ',S);
    end;
    Writeln ('Stopped conversation');
    Close (Si);
    Writeln ('Put down phone');
    end
Else
begin
    Writeln ('This is the son ');
    While not eof (input) do
        begin
            readln (s);
            if pos ('Hello son !',S)<>0 then
                Writeln ('Hello Dad !')
            else
                writeln ('Who are you ?');
            end;
        close (output);
        end
    end.

```

81.3.3 EpochToLocal

Declaration: `procedure EpochToLocal (epoch: Int64; var year: Word; var month: Word; var day: Word; var hour: Word; var minute: Word; var second: Word)`

Visibility: default

81.3.4 EpochToUniversal

Declaration: `procedure EpochToUniversal (epoch: Int64; var year: Word; var month: Word; var day: Word; var hour: Word; var minute: Word; var second: Word)`

Visibility: default

81.3.5 FpExecL

Synopsis: Execute process (using argument list, environment).

Declaration: `function FpExecL (const PathName: RawByteString; const S: Array of RawByteString) : cint`

Visibility: default

Description: `FpExecL` replaces the currently running program with the program, specified in `PathName`. `S` is an array of command options. The executable in `PathName` must be an absolute pathname. The current process' environment is passed to the program. On success, `FpExecL` does not return.

Errors: Extended error information is returned by the `FpGetErrno` ([207](#)) function:

sys_eaccess File is not a regular file, or has no execute permission. A component of the path has no search permission.

sys_eperm The file system is mounted *noexec*.
sys_e2big Argument list too big.
sys_enoexec The magic number in the file is incorrect.
sys_enoent The file does not exist.
sys_enomem Not enough memory for kernel, or to split command line.
sys_enotdir A component of the path is not a directory.
sys_eloop The path contains a circular reference (via symlinks).

See also: `FpExecve` (201), `FpExecv` (2210), `FpExecvp` (2211), `FpExecl` (2208), `FpExeclp` (2209), `FpFork` (204)

Listing: `./examples/unixex/ex77.pp`

Program `Example77;`

{ Program to demonstrate the FPExecl function. }

Uses `Unix, strings;`

begin
{ Execute 'ls -l', with current environment. }
{ 'ls' is NOT looked for in PATH environment variable. }
`FpExecl ('/bin/ls', ['-l']);`
end.

81.3.6 FpExecLE

Synopsis: Execute process (using argument list, environment).

Declaration: `function FpExecLE(const PathName: RawByteString;
const S: Array of RawByteString; MyEnv: PPChar) : cint`

Visibility: `default`

Description: `FpExecLE` replaces the currently running program with the program, specified in `PathName`. `S` is an array of command options. The executable in `PathName` must be an absolute pathname. The environment in `MyEnv` is passed to the program. On success, `FpExecLE` does not return.

Errors: Extended error information is returned by the `FpGetErrno` (207) function:

sys_eaccess File is not a regular file, or has no execute permission. A component of the path has no search permission.
sys_eperm The file system is mounted *noexec*.
sys_e2big Argument list too big.
sys_enoexec The magic number in the file is incorrect.
sys_enoent The file does not exist.
sys_enomem Not enough memory for kernel, or to split command line.
sys_enotdir A component of the path is not a directory.
sys_eloop The path contains a circular reference (via symlinks).

See also: `FpExecve` (201), `FpExecv` (2210), `FpExecvp` (2211), `FpExecl` (2207), `FpExeclp` (2209), `FpFork` (204)

Listing: ./examples/unixex/ex11.pp

Program Example11;

{ Program to demonstrate the Execle function. }

Uses Unix, strings;

begin

{ Execute 'ls -l', with current environment. }
{ 'ls' is NOT looked for in PATH environment variable. }
{ envp is defined in the system unit. }
 Execle ('/bin/ls -l',envp);

end.

81.3.7 FpExecLP

Synopsis: Execute process (using argument list, environment; search path).

Declaration: `function FpExecLP(const PathName: RawByteString;
 const S: Array of RawByteString) : cint`

Visibility: default

Description: FpExecLP replaces the currently running program with the program, specified in PathName. S is an array of command options. The executable in PathName is searched in the path, if it isn't an absolute filename. The current environment is passed to the program. On success, FpExecLP does not return.

Errors: Extended error information is returned by the FpGetErrno ([207](#)) function:

sys_eaccess File is not a regular file, or has no execute permission. A component of the path has no search permission.

sys_eperm The file system is mounted *noexec*.

sys_e2big Argument list too big.

sys_enoexec The magic number in the file is incorrect.

sys_enoent The file does not exist.

sys_enomem Not enough memory for kernel, or to split command line.

sys_enotdir A component of the path is not a directory.

sys_eloop The path contains a circular reference (via symlinks).

See also: FpExecve ([201](#)), FpExecv ([2210](#)), FpExecvp ([2211](#)), FpExecle ([2208](#)), FpExecl ([2207](#)), FpFork ([204](#))

Listing: ./examples/unixex/ex76.pp

Program Example76;

{ Program to demonstrate the FpExeclp function. }

Uses Unix, strings;

begin

{ Execute 'ls -l', with current environment. }

```

    { 'ls' is looked for in PATH environment variable.}
    { envp is defined in the system unit.}
    FpExeclp ('ls', ['-l']);
end.

```

81.3.8 FpExecLPE

Synopsis: Execute a program in the path, and pass it an environment.

Declaration: `function FpExecLPE(const PathName: RawByteString;
const S: Array of RawByteString; env: PPChar) : cint`

Visibility: default

Description: `FpExecLPE` does the same as `FpExecLP` (2209), but additionally it specifies the environment for the new process in `env`, a pointer to a null-terminated array of null-terminated strings.

Errors: On success, this function does not return.

See also: `FpExecLP` (2209), `FpExecLE` (2208)

81.3.9 FpExecV

Synopsis: Execute process.

Declaration: `function FpExecV(const PathName: RawByteString; args: PPChar) : cint`

Visibility: default

Description: `FpExecV` replaces the currently running program with the program, specified in `PathName`. It gives the program the options in `args`. This is a pointer to an array of pointers to null-terminated strings. The last pointer in this array should be nil. The current environment is passed to the program. On success, `FpExecV` does not return.

Errors: Extended error information is returned by the `FpGetErrno` (207) function:

sys_eaccess File is not a regular file, or has no execute permission. A component of the path has no search permission.

sys_eperm The file system is mounted *noexec*.

sys_e2big Argument list too big.

sys_enoexec The magic number in the file is incorrect.

sys_enoent The file does not exist.

sys_enomem Not enough memory for kernel.

sys_enotdir A component of the path is not a directory.

sys_eloop The path contains a circular reference (via symlinks).

See also: `FpExecve` (201), `FpExecvp` (2211), `FpExecl` (2208), `FpExeccl` (2207), `FpExeclp` (2209), `FpFork` (204)

Listing: `./examples/unixex/ex8.pp`

Program Example8;

{ Program to demonstrate the Execv function. }

Uses Unix, strings;

Const Arg0 : PChar = '/bin/l\$';
 Arg1 : Pchar = '-l';

Var PP : PPchar;

begin

GetMem (PP,3*SizeOf(Pchar));
 PP[0]:=Arg0;
 PP[1]:=Arg1;
 PP[3]:=Nil;
 { Execute '/bin/l\$ -l', with current environment }
 fpExecv ('/bin/l\$',pp);

end.

81.3.10 FpExecVP

Synopsis: Execute process, search path.

Declaration: function FpExecVP(const PathName: RawByteString; args: PPChar) : cint

Visibility: default

Description: FpExecVP replaces the currently running program with the program, specified in PathName. The executable in path is searched in the path, if it isn't an absolute filename. It gives the program the options in args. This is a pointer to an array of pointers to null-terminated strings. The last pointer in this array should be nil. The current environment is passed to the program. On success, `execvp` does not return.

Errors: Extended error information is returned by the FpGetErrno ([207](#)) function:

sys_eaccess File is not a regular file, or has no execute permission. A component of the path has no search permission.

sys_eperm The file system is mounted *noexec*.

sys_e2big Argument list too big.

sys_enoexec The magic number in the file is incorrect.

sys_enoent The file does not exist.

sys_enomem Not enough memory for kernel.

sys_enotdir A component of the path is not a directory.

sys_eloop The path contains a circular reference (via symlinks).

See also: FpExecve ([201](#)), FpExecv ([2210](#)), FpExecl ([2208](#)), FpExecl ([2207](#)), FpExeclp ([2209](#)), FpFork ([204](#))

Listing: ./examples/unixex/ex79.pp

Program Example79;

{ Program to demonstrate the FpExecVP function. }

Uses Unix, strings;

Const Arg0 : PChar = 'ls';
 Arg1 : Pchar = '-l';

Var PP : PPchar;

begin

GetMem (PP,3*SizeOf(Pchar));

 PP[0]:=Arg0;

 PP[1]:=Arg1;

 PP[2]:=Nil;

{ Execute 'ls -l', with current environment. }

{ 'ls' is looked for in PATH environment variable. }

 fpExecvp ('ls',pp);

end.

81.3.11 FpExecVPE

Synopsis: Execute process, search path using environment.

Declaration: function FpExecVPE(const PathName: RawByteString; args: PPChar;
 env: PPChar) : cint

Visibility: default

Description: FpExecVP replaces the currently running program with the program, specified in PathName. The executable in path is searched in the path, if it isn't an absolute filename. It gives the program the options in args. This is a pointer to an array of pointers to null-terminated strings. The last pointer in this array should be nil. The environment in Env is passed to the program. On success, execvp does not return.

Errors: Extended error information is returned by the FpGetErrno ([207](#)) function:

sys_eaccess File is not a regular file, or has no execute permission. A component of the path has no search permission.

sys_eperm The file system is mounted *noexec*.

sys_e2big Argument list too big.

sys_enoexec The magic number in the file is incorrect.

sys_enoent The file does not exist.

sys_enomem Not enough memory for kernel.

sys_enotdir A component of the path is not a directory.

sys_eloop The path contains a circular reference (via symlinks).

See also: FpExecve ([201](#)), FpExecv ([2210](#)), FpExecle ([2208](#)), FpExecl ([2207](#)), FpExeclp ([2209](#)), FpFork ([204](#))

Listing: ./examples/unixex/ex79.pp

Program Example79;

{ Program to demonstrate the FpExecVP function. }

Uses Unix, strings;

Const Arg0 : PChar = 'ls';
 Arg1 : Pchar = '-l';

Var PP : PPchar;

begin

GetMem (PP,3*SizeOf(Pchar));

 PP[0]:=Arg0;

 PP[1]:=Arg1;

 PP[2]:=Nil;

{ Execute 'ls -l', with current environment. }

{ 'ls' is looked for in PATH environment variable. }

 fpExecvp ('ls',pp);

end.

81.3.12 fpFlock

Synopsis: Lock a file (advisory lock).

Declaration: function fpFlock(var T: text; mode: cint) : cint
 function fpFlock(var F: File; mode: cint) : cint
 function fpFlock(fd: cint; mode: cint) : cint

Visibility: default

Description: FpFlock implements file locking. it sets or removes a lock on the file F. F can be of type Text or File, or it can be a Linux file descriptor (a longint) Mode can be one of the following constants :

LOCK_SH sets a shared lock.

LOCK_EX sets an exclusive lock.

LOCK_UN unlocks the file.

LOCK_NB This can be OR-ed together with the other. If this is done the application doesn't block when locking.

The function returns zero if successful, a nonzero return value indicates an error.

Errors: Extended error information is returned by the FpGetErrno ([207](#)) function:

See also: #rtl.baseunix.FpFcntl ([202](#)), fpFSync ([2213](#))

81.3.13 fpfStatFS

Synopsis: Retrieve file system information.

Declaration: function fpfStatFS(Fd: cint; Info: pstatfs) : cint

Visibility: default

Description: `fpStatFS` returns in `Info` information about the file system on which the open file descriptor `fd` resides. `Info` is of type `tstatfs`. The function returns 0 if the call was successful, or an error code if the call failed.

Errors: On error, a non-zero error code is returned

See also: `fpStatFS` (2214), `#rtl.baseunix.fpfStat` (204)

81.3.14 `fpfsync`

Synopsis: Synchronize file's kernel data with disk.

Declaration: `function fpfsync(fd: cint) : cint`

Visibility: default

Description: `fpFSync` synchronizes the kernel data for file `fd` (the cache) with the disk. The call will not return till all file data was written to disk.

If the call was successful, 0 is returned. On failure, a nonzero value is returned.

Errors: Extended error information is returned by the `FpGetErrno` (207) function:

See also: `FpFLock` (2213)

81.3.15 `fpgettimeofday`

Synopsis: Return kernel time of day in GMT.

Declaration: `function fpgettimeofday(tp: ptimeval; tzp: ptimezone) : cint`

Visibility: default

Description: `FpGetTimeOfDay` returns the number of seconds since 00:00, January 1 1970, GMT in a `timeval` record. This time NOT corrected any way, not taking into account timezones, daylight savings time and so on.

It is simply a wrapper to the kernel system call.

Errors: None.

81.3.16 `fpStatFS`

Synopsis: Retrieve file system information.

Declaration: `function fpStatFS(Path: PChar; Info: pstatfs) : cint`
`function fpStatFS(Path: ansistring; Info: pstatfs) : cint`

Visibility: default

Description: `fpStatFS` returns in `Info` information about the file system on which the file or path `Path` resides. `Info` is of type `tstatfs`. The function returns 0 if the call was successful, or an error code if the call failed.

Errors: On error, a non-zero error code is returned

See also: `fpFStatFS` (2213), `#rtl.baseunix.fpfStat` (237)

81.3.17 fpSystem

Synopsis: Execute and feed command to system shell.

Declaration: `function fpSystem(const Command: RawByteString) : cint`

Visibility: default

Description: `FpSystem` invokes the bash shell (`/bin/sh`), and feeds it the command `Command` (using the `-c` option). The function then waits for the command to complete, and then returns the exit status of the command in `wait(3)` format, or 127 if it could not complete the `FpFork` (204) or `FpExecve` (201) calls. To convert the return value of `fpssystem` to the real return value use `WEXITSTATUS()` (246).

Errors: Errors are reported in `(fpget)ErrNo` (207)

See also: `POpen` (2219), `FpFork` (204), `FpExecve` (201)

Listing: `./examples/unixex/ex80.pp`

```

program example56;

uses Unix;

{ Program to demonstrate the Shell function }

Var S : Longint;

begin
  Writeln ('Output of ls -l *.pp');
  S:=fpSystem('ls -l *.pp');
  Writeln ('Command exited with status : ',S);
end.

```

81.3.18 FSearch

Synopsis: Search for file in search path.

Declaration: `function FSearch(const path: RawByteString; dirlist: RawByteString; CurrentDirStrategy: TFSearchOption) : RawByteString`
`function FSearch(const path: RawByteString; dirlist: RawByteString) : RawByteString`
`function FSearch(const path: UnicodeString; dirlist: UnicodeString; CurrentDirStrategy: TFSearchOption) : UnicodeString`
`function FSearch(const path: UnicodeString; dirlist: UnicodeString) : UnicodeString`

Visibility: default

Description: `FSearch` searches in `DirList`, a colon separated list of directories, for a file named `Path`. It then returns a path to the found file.

The `CurrentDirStrategy` determines how the current directory is treated when searching:

NoCurrentDirectory Do not search the current directory unless it is specified in the search path.

CurrentDirectoryFirstSearch the current directory first, before all directories in the search path.

CurrentDirectoryLastSearch the current directory last, after all directories in the search path.

It is mainly provided to mimic DOS search path behaviour. Default behaviour is to search the current directory first.

Errors: An empty string if no such file was found.

See also: `#rtl.unixutil.FNMatch` ([2245](#))

Listing: `./examples/unixex/ex46.pp`

Program Example46;

{ Program to demonstrate the FSearch function. }

Uses BaseUnix, Unix, Strings;

begin

WriteLn ('Is in : ', FSearch ('ls', **strpas**(fpGetenv('PATH'))));
end.

81.3.19 GetDomainName

Synopsis: Return current domain name.

Declaration: `function GetDomainName : string`

Visibility: default

Description: Get the domain name of the machine on which the process is running. An empty string is returned if the domain is not set.

Errors: None.

See also: `GetHostName` ([2216](#))

Listing: `./examples/unixex/ex39.pp`

Program Example39;

{ Program to demonstrate the GetDomainName function. }

Uses Unix;

begin

WriteLn ('Domain name of this machine is : ', GetDomainName);
end.

81.3.20 GetHostName

Synopsis: Return host name.

Declaration: `function GetHostName : string`

Visibility: default

Description: Get the hostname of the machine on which the process is running. An empty string is returned if hostname is not set.

Errors: None.

See also: `GetDomainName` ([2216](#))

Listing: `./examples/unixex/ex40.pp`

Program Example40;

{ Program to demonstrate the GetHostName function. }

Uses unix;

begin

Writeln ('Name of this machine is : ', GetHostName);

end.

81.3.21 GetLocalTimezone

Synopsis: Return local timezone information.

Declaration:

```
function GetLocalTimezone(timer: Int64; timerIsUTC: Boolean;
    var ATZInfo: TTZInfo; var ATZInfoEx: TTZInfoEx)
    : Boolean
function GetLocalTimezone(timer: Int64; timerIsUTC: Boolean;
    var ATZInfo: TTZInfo) : Boolean
```

Visibility: default

Description: `GetLocalTimezone` returns the local timezone information. It also initializes the `TZSeconds` variable, which is used to correct the epoch time to local time.

There should never be any need to call this function directly. It is called by the initialization routines of the Linux unit.

See also: `GetTimezoneFile` ([2217](#)), `ReadTimezoneFile` ([2220](#))

81.3.22 GetTimezoneFile

Synopsis: Return name of timezone information file.

Declaration: `function GetTimezoneFile: string`

Visibility: default

Description: `GetTimezoneFile` returns the location of the current timezone file. The location of file is determined as follows:

- 1.If `/etc/timezone` exists, it is read, and the contents of this file is returned. This should work on Debian systems.
- 2.If `/usr/lib/zoneinfo/localtime` exists, then it is returned. (this file is a symlink to the timezone file on SuSE systems)
- 3.If `/etc/localtime` exists, then it is returned. (this file is a symlink to the timezone file on RedHat systems)

Errors: If no file was found, an empty string is returned.

See also: `ReadTimezoneFile` ([2220](#))

81.3.23 Gettzdaylight

Declaration: `function Gettzdaylight : Boolean`

Visibility: default

81.3.24 GetTZInfo

Declaration: `function GetTZInfo : TTZInfo`

Visibility: default

81.3.25 GetTZInfoEx

Declaration: `function GetTZInfoEx : TTZInfoEx`

Visibility: default

81.3.26 Gettzname

Declaration: `function Gettzname(const b: Boolean) : string`

Visibility: default

81.3.27 GetTzseconds

Declaration: `function GetTzseconds : LongInt`

Visibility: default

81.3.28 GregorianToJulian

Declaration: `function GregorianToJulian(Year: LongInt; Month: LongInt; Day: LongInt)
: LongInt`

Visibility: default

81.3.29 JulianToGregorian

Declaration: `procedure JulianToGregorian(JulianDN: LongInt; var Year: Word;
var Month: Word; var Day: Word)`

Visibility: default

81.3.30 LocalToEpoch

Declaration: `function LocalToEpoch(year: Word; month: Word; day: Word; hour: Word;
minute: Word; second: Word) : Int64`

Visibility: default

81.3.31 PClose

Synopsis: Close file opened with POpen (2219).

Declaration: `function PClose(var F: File) : cint`
`function PClose(var F: text) : cint`

Visibility: default

Description: PClose closes a file opened with POpen (2219). It waits for the command to complete, and then returns the exit status of the command.

For an example, see POpen (2219)

Errors: Extended error information is returned by the FpGetErrno (207) function.

See also: POpen (2219)

81.3.32 POpen

Synopsis: Pipe file to standard input/output of program.

Declaration: `function POpen(var F: text; const Prog: RawByteString; rw: Char) : cint`
`function POpen(var F: File; const Prog: RawByteString; rw: Char) : cint`
`function POpen(var F: text; const Prog: UnicodeString; rw: Char) : cint`
`function POpen(var F: File; const Prog: UnicodeString; rw: Char) : cint`

Visibility: default

Description: POpen runs the command specified in Prog, and redirects the standard in or output of the command to the other end of the pipe F. The parameter rw indicates the direction of the pipe. If it is set to 'W', then F can be used to write data, which will then be read by the command from stdin. If it is set to 'R', then the standard output of the command can be read from F. F should be reset or rewritten prior to using it. F can be of type Text or File. A file opened with POpen can be closed with Close, but also with PClose (2218). The result is the same, but PClose returns the exit status of the command Prog.

Errors: Extended error information is returned by the FpGetErrno (207) function. Errors are essentially those of the Execve, Dup and AssignPipe commands.

See also: AssignPipe (2204), PClose (2218)

Listing: ./examples/unixex/ex37.pp

Program Example37;

{ Program to demonstrate the Popen function. }

uses BaseUnix, Unix;

var f : text;
i : longint;

begin

writeln ('Creating a shell script to which echoes its arguments');
writeln ('and input back to stdout');
assign (f, 'test21a');
rewrite (f);
writeln (f, '#!/bin/sh');

```

writeln (f,'echo this is the child speaking.... ');
writeln (f,'echo got arguments \*$*\n');
writeln (f,'cat');
writeln (f,'exit 2');
writeln (f);
close (f);
fpchmod ('test21a',&755);
popen (f,'./test21a arg1 arg2','W');
if fpgeterrno<>0 then
    writeln ('error from POpen : errno : ', fpgeterrno);
for i:=1 to 10 do
    writeln (f,'This is written to the pipe, and should appear on stdout. ');
Flush(f);
Writeln ('The script exited with status : ',PClose (f));
writeln;
writeln ('Press <return> to remove shell script. ');
readln;
assign (f,'test21a');
erase (f)
end.

```

81.3.33 ReadTimezoneFile

Synopsis: Read the timezone file and initialize time routines.

Declaration: `function ReadTimezoneFile(fn: string) : Boolean`

Visibility: default

Description: `ReadTimezoneFile` reads the timezone file `fn` and initializes the local time routines based on the information found there.

There should be no need to call this function. The initialization routines of the linux unit call this routine at unit startup.

Errors: None.

See also: `GetTimezoneFile` ([2217](#)), `GetLocalTimezone` ([2217](#))

81.3.34 RefreshTZInfo

Declaration: `procedure RefreshTZInfo`

Visibility: default

81.3.35 ReReadLocalTime

Synopsis: Re-Read the local time files.

Declaration: `procedure ReReadLocalTime`

Visibility: default

Description: `ReReadLocalTime` can be used to re-initialize the local timezone information.

To speed up conversion of epoch (UTC) time to local time, the timezone information is loaded only once, at program startup. Calling this routine re-reads the timezone information using current timezone settings.

The `EpochToLocal` (2245) function uses timezone information to transform epoch time to local time. This timezone information does not change while the application is running: in particular, on DST transitions or when the timezone files change, the time returned by local time routines will be wrong.

See also: `Date` (1673), `Time` (1781), `Now` (1745), `EpochToLocal` (2245)

81.3.36 `SeekDir`

Synopsis: Seek to position in directory.

Declaration: `procedure SeekDir(p: pDir; loc: clong)`

Visibility: default

Description: `SeekDir` sets the directory pointer to the `loc`-th entry in the directory structure pointed to by `p`.

For an example, see `#rtl.baseunix.fpOpenDir` (221).

Errors: Extended error information is returned by the `FpGetErrno` (207) function:

See also: `#rtl.baseunix.fpCloseDir` (198), `#rtl.baseunix.fpReadDir` (225), `#rtl.baseunix.fpOpenDir` (221), `TellDir` (2221)

81.3.37 `SetTZInfo`

Declaration: `procedure SetTZInfo(const ATZInfo: TTZInfo; const ATZInfoEx: TTZInfoEx)`

Visibility: default

81.3.38 `TellDir`

Synopsis: Return current location in a directory.

Declaration: `function TellDir(p: pDir) : TOff`

Visibility: default

Description: `TellDir` returns the current location in the directory structure pointed to by `p`. It returns -1 on failure.

For an example, see `#rtl.baseunix.fpOpenDir` (221).

See also: `#rtl.baseunix.fpCloseDir` (198), `#rtl.baseunix.fpReadDir` (225), `#rtl.baseunix.fpOpenDir` (221), `SeekDir` (2220)

81.3.39 `UniversalToEpoch`

Declaration: `function UniversalToEpoch(year: Word; month: Word; day: Word;
hour: Word; minute: Word; second: Word) : Int64`

Visibility: default

81.3.40 WaitProcess

Synopsis: Wait for process to terminate.

Declaration: `function WaitProcess(Pid: cint) : cint`

Visibility: default

Description: `WaitProcess` waits for process `PID` to exit. `WaitProcess` is equivalent to the `#rtl.baseunix.FpWaitPID` (245) call:

```
FpWaitPid(PID, @result, 0)
```

Handles of Signal interrupts (`errno=EINTR`), and returns the Exitcode of Process `PID` (≥ 0) or - Status if it was terminated

Errors: None.

See also: `#rtl.baseunix.FpWaitPID` (245), `#rtl.baseunix.WTERMSIG` (247), `#rtl.baseunix.WSTOPSIG` (247), `#rtl.baseunix.WIFEXITED` (246), `WIFSTOPPED` (222), `#rtl.baseunix.WIFSIGNALED` (247), `W_EXITCODE` (222), `W_STOPCODE` (222), `#rtl.baseunix.WEXITSTATUS` (246)

81.3.41 WIFSTOPPED

Synopsis: Check whether the process is currently stopped.

Declaration: `function WIFSTOPPED(Status: Integer) : Boolean`

Visibility: default

Description: `WIFSTOPPED` checks `Status` and returns `true` if the process is currently stopped. This is only possible if `WUNTRACED` was specified in the options of `FpWaitPID` (245).

See also: `#rtl.baseunix.FpWaitPID` (245), `WaitProcess` (221), `#rtl.baseunix.WTERMSIG` (247), `#rtl.baseunix.WSTOPSIG` (247), `#rtl.baseunix.WIFEXITED` (246), `#rtl.baseunix.WIFSIGNALED` (247), `W_EXITCODE` (222), `W_STOPCODE` (222), `#rtl.baseunix.WEXITSTATUS` (246)

81.3.42 W_EXITCODE

Synopsis: Construct an exit status based on an return code and signal.

Declaration: `function W_EXITCODE(ReturnCode: Integer; Signal: Integer) : Integer`

Visibility: default

Description: `W_EXITCODE` combines `ReturnCode` and `Signal` to a status code fit for `WaitPid`.

See also: `#rtl.baseunix.FpWaitPID` (245), `WaitProcess` (221), `#rtl.baseunix.WTERMSIG` (247), `#rtl.baseunix.WSTOPSIG` (247), `#rtl.baseunix.WIFEXITED` (246), `WIFSTOPPED` (222), `#rtl.baseunix.WIFSIGNALED` (247), `W_EXITCODE` (222), `W_STOPCODE` (222), `#rtl.baseunix.WEXITSTATUS` (246)

81.3.43 W_STOPCODE

Synopsis: Construct an exit status based on a signal.

Declaration: `function W_STOPCODE(Signal: Integer) : Integer`

Visibility: default

Description: `W_STOPCODE` constructs an exit status based on `Signal`, which will cause `WIFSIGNALED` (247) to return `True`

See also: `#rtl.baseunix.FpWaitPID` (245), `WaitProcess` (2221), `#rtl.baseunix.WTERMSIG` (247), `#rtl.baseunix.WSTOPSIG` (247), `#rtl.baseunix.WIFEXITED` (246), `WIFSTOPPED` (2222), `#rtl.baseunix.WIFSIGNALED` (247), `W_EXITCODE` (2222), `#rtl.baseunix.WEXITSTATUS` (246)

81.4 TTZInfo

```
TTZInfo = record
  daylight : Boolean;
  seconds : LongInt;
  validsince
    : Int64;
  validuntil : Int64;
end
```

81.5 TTZInfoEx

```
TTZInfoEx = record
  name : Array[boolean] of RawByteString;
  leap_correct
    : LongInt;
  leap_hit : LongInt;
end
```


Chapter 82

Reference for unit 'unixcp'

82.1 Used units

Table 82.1: Used units by unit 'unixcp'

Name	Page
BaseUnix	155
System	1335

82.2 Overview

The `unixcp` unit provides routines to handle mapping of code page names to numerical values as used in `libiconv`. The `GetCodepageByName` ([2227](#)) function is the main function for this. The `GetCodepageData` ([2228](#)) can be used to map a code page number to a name. These function can be used for instance to map code page information in environment variables to code page numbers used in string encodings. The supported code page names are the ones commonly in use in `libiconv`.

This unit is used for example in unit `cwstring` ([608](#)).

82.3 Constants, types and variables

82.3.1 Constants

```
UnixCpMap : Array[-1..UnixCpMapLimit] of TUnixCpData = ((cp: 0; name
: 'UTF-8'), (cp: 37; name: 'IBM037'), (cp: 37; name: 'IBM-037'),
(cp: 154; name: 'CP154'), (cp: 154; name: 'CYRILLIC-ASIAN'), (cp:
154; name: 'PT154'), (cp: 154; name: 'PTCP154'), (cp: 154; name:
'CSPTCP154'), (cp: 437; name: '437'), (cp: 437; name: 'CP437'),
(cp: 437; name: 'IBM-437'), (cp: 437; name: 'CSPC8CODEPAGE437'),
(cp: 437; name: 'IBM437'), (cp: 500; name: 'IBM500'), (cp: 500; name
: 'IBM-500'), (cp: 708; name: 'ASMO-708'), (cp: 720; name: 'DOS-720'
), (cp: 737; name: 'CP737'), (cp: 737; name: 'ibm737'), (cp: 775;
name: 'CP775'), (cp: 775; name: 'IBM775'), (cp: 775; name: 'CSPC775BALTIC'
), (cp: 775; name: 'ibm775'), (cp: 850; name: '850'), (cp: 850; name
: 'CP850'), (cp: 850; name: 'IBM850'), (cp: 850; name: 'CSPC850MULTILINGUAL'
```

```

), (cp: 850; name: 'ibm850'), (cp: 852; name: '852'), (cp: 852; name
: 'CP852'), (cp: 852; name: 'IBM852'), (cp: 852; name: 'CSPCP852'
), (cp: 852; name: 'ibm852'), (cp: 853; name: 'CP853'), (cp: 855;
name: '855'), (cp: 855; name: 'CP855'), (cp: 855; name: 'IBM855'
), (cp: 855; name: 'CSIBM855'), (cp: 855; name: 'IBM855'), (cp: 857
; name: '857'), (cp: 857; name: 'CP857'), (cp: 857; name: 'IBM857'
), (cp: 857; name: 'CSIBM857'), (cp: 857; name: 'ibm857'), (cp: 858
; name: 'CP858'), (cp: 858; name: 'IBM00858'), (cp: 860; name: '860'
), (cp: 860; name: 'CP860'), (cp: 860; name: 'IBM860'), (cp: 860;
name: 'CSIBM860'), (cp: 860; name: 'IBM860'), (cp: 861; name: '861'
), (cp: 861; name: 'CP-IS'), (cp: 861; name: 'CP861'), (cp: 861; name
: 'IBM861'), (cp: 861; name: 'CSIBM861'), (cp: 861; name: 'ibm861'
), (cp: 862; name: '862'), (cp: 862; name: 'CP862'), (cp: 862; name
: 'IBM862'), (cp: 862; name: 'CSPC862LATINHEBREW'), (cp: 862; name
: 'DOS-862'), (cp: 863; name: '863'), (cp: 863; name: 'CP863'), (cp
: 863; name: 'CSIBM863'), (cp: 863; name: 'IBM863'), (cp: 864; name
: 'CP864'), (cp: 864; name: 'CSIBM864'), (cp: 864; name: 'IBM864'
), (cp: 865; name: '865'), (cp: 865; name: 'IBM-865'), (cp: 865; name
: 'CP865'), (cp: 865; name: 'CSIBM865'), (cp: 865; name: 'IBM865'
), (cp: 866; name: '866'), (cp: 866; name: 'CP866'), (cp: 866; name
: 'IBM866'), (cp: 866; name: 'CSIBM866'), (cp: 866; name: 'cp866'
), (cp: 869; name: '869'), (cp: 869; name: 'IBM-869'), (cp: 869; name
: 'CP-GR'), (cp: 869; name: 'CP869'), (cp: 869; name: 'IBM869'),
(cp: 869; name: 'CSIBM869'), (cp: 869; name: 'ibm869'), (cp: 870;
name: 'IBM870'), (cp: 874; name: 'CP874'), (cp: 874; name: 'WINDOWS-874'
), (cp: 874; name: 'windows-874'), (cp: 875; name: 'cp875'), (cp:
932; name: 'CP932'), (cp: 932; name: 'IBM-943'), (cp: 932; name:
'MS932'), (cp: 932; name: 'SHIFFT_JIS'), (cp: 932; name: 'SHIFFT_JIS-MS'
), (cp: 932; name: 'SJIS'), (cp: 932; name: 'SJIS-MS'), (cp: 932;
name: 'SJIS-OPEN'), (cp: 932; name: 'SJIS-WIN'), (cp: 932; name:
'WINDOWS-31J'), (cp: 932; name: 'WINDOWS-932'), (cp: 932; name: 'CSWINDOWS31J'
), (cp: 932; name: 'shift_jis'), (cp: 932; name: 'shift-jis'), (cp
: 936; name: 'CP936'), (cp: 936; name: 'GBK'), (cp: 936; name: 'MS936'
), (cp: 936; name: 'WINDOWS-936'), (cp: 936; name: 'gb2312'), (cp
: 949; name: 'CP949'), (cp: 949; name: 'UHC'), (cp: 949; name: 'EUC-KR'
), (cp: 949; name: 'ks_c_5601-1987'), (cp: 950; name: 'CP950'), (cp
: 950; name: 'BIG5'), (cp: 950; name: 'big5'), (cp: 1026; name: 'IBM1026'
), (cp: 1047; name: 'IBM01047'), (cp: 1125; name: 'CP1125'), (cp:
1125; name: 'IBM-1125'), (cp: 1133; name: 'CP1133'), (cp: 1133; name
: 'IBM-1133'), (cp: 1133; name: 'IBM-CP1133'), (cp: 1140; name: 'IBM01140'
), (cp: 1141; name: 'IBM01141'), (cp: 1142; name: 'IBM01142'), (cp
: 1143; name: 'IBM01143'), (cp: 1144; name: 'IBM01144'), (cp: 1145
; name: 'IBM01145'), (cp: 1146; name: 'IBM01146'), (cp: 1147; name
: 'IBM01147'), (cp: 1148; name: 'IBM01148'), (cp: 1149; name: 'IBM01149'
), (cp: 1200; name: 'UTF-16LE'), (cp: 1200; name: 'UTF16LE'), (cp
: 1200; name: 'UCS-2LE'), (cp: 1200; name: 'CP1200'), (cp: 1200; name
: 'UTF16'), (cp: 1200; name: 'UTF-16'), (cp: 1200; name: 'UCS-2')
, (cp: 1201; name: 'UTF-16BE'), (cp: 1201; name: 'UTF16BE'), (cp:
1201; name: 'UCS-2BE'), (cp: 1201; name: 'unicodeFFFE'), (cp: 1201
; name: 'CP1201'), (cp: 1250; name: 'CP1250'), (cp: 1250; name: 'MS-EE'
), (cp: 1250; name: 'WINDOWS-1250'), (cp: 1250; name: 'windows-1250'
), (cp: 1251; name: 'CP1251'), (cp: 1251; name: 'MS-CYRL'), (cp: 1251
; name: 'WINDOWS-1251'), (cp: 1251; name: 'windows-1251'), (cp: 1252
; name: 'CP1252'), (cp: 1252; name: 'MS-ANSI'), (cp: 1252; name: 'WINDOWS-1252'

```

```

), (cp: 1252; name: 'windows-1252'), (cp: 1253; name: 'CP1253'),
(cp: 1253; name: 'MS-GREEK'), (cp: 1253; name: 'WINDOWS-1253'), (cp
: 1253; name: 'windows-1253'), (cp: 1254; name: 'CP1254'), (cp: 1254
; name: 'MS-TURK'), (cp: 1254; name: 'WINDOWS-1254'), (cp: 1254; name
: 'windows-1254'), (cp: 1255; name: 'CP1255'), (cp: 1255; name: 'MS-HEBR'
), (cp: 1255; name: 'WINDOWS-1255'), (cp: 1255; name: 'windows-1255'
), (cp: 1256; name: 'CP1256'), (cp: 1256; name: 'MS-ARAB'), (cp: 1256
; name: 'WINDOWS-1256'), (cp: 1256; name: 'windows-1256'), (cp: 1257
; name: 'CP1257'), (cp: 1257; name: 'WINBALTRIM'), (cp: 1257; name
: 'WINDOWS-1257'), (cp: 1257; name: 'windows-1257'), (cp: 1258; name
: 'CP1258'), (cp: 1258; name: 'WINDOWS-1258'), (cp: 1258; name: 'windows-1258'
), (cp: 1361; name: 'CP1361'), (cp: 1361; name: 'JOHAB'), (cp: 1361
; name: 'Johab'), (cp: 10000; name: 'macintosh'), (cp: 10001; name
: 'x-mac-japanese'), (cp: 10002; name: 'x-mac-chinesetrad'), (cp:
10003; name: 'x-mac-korean'), (cp: 10004; name: 'x-mac-arabic'),
(cp: 10005; name: 'x-mac-hebrew'), (cp: 10006; name: 'x-mac-greek'
), (cp: 10007; name: 'x-mac-cyrillic'), (cp: 10008; name: 'x-mac-chinesesimp'
), (cp: 10010; name: 'x-mac-romanian'), (cp: 10017; name: 'x-mac-ukrainian'
), (cp: 10021; name: 'x-mac-thai'), (cp: 10029; name: 'x-mac-ce')
, (cp: 10079; name: 'x-mac-icelandic'), (cp: 10081; name: 'x-mac-turkish'
), (cp: 10082; name: 'x-mac-croatian'), (cp: 12000; name: 'UTF-32LE'
), (cp: 12000; name: 'CP12000'), (cp: 12000; name: 'UTF32LE'), (cp
: 12000; name: 'UTF32'), (cp: 12000; name: 'UTF-32'), (cp: 12001;
name: 'UTF-32BE'), (cp: 12001; name: 'CP12001'), (cp: 12001; name
: 'UTF32BE'), (cp: 20000; name: 'x-Chinese_CNS'), (cp: 20001; name
: 'x-cp20001'), (cp: 20002; name: 'x_Chinese-Eten'), (cp: 20003; name
: 'x-cp20003'), (cp: 20004; name: 'x-cp20004'), (cp: 20005; name:
'x-cp20005'), (cp: 20105; name: 'x-IA5'), (cp: 20106; name: 'x-IA5-German'
), (cp: 20107; name: 'x-IA5-Swedish'), (cp: 20108; name: 'x-IA5-Norwegian'
), (cp: 20127; name: 'US-ASCII'), (cp: 20127; name: 'ASCII'), (cp
: 20127; name: 'ANSI_X3.4-1968'), (cp: 20127; name: 'ANSI_X3.4-1986'
), (cp: 20127; name: 'CP367'), (cp: 20127; name: 'IBM367'), (cp: 20127
; name: 'ISO-IR-6'), (cp: 20127; name: 'ISO646-US'), (cp: 20127; name
: 'ISO_646.IRV:1991'), (cp: 20127; name: 'US'), (cp: 20127; name:
'CSASCII'), (cp: 20127; name: 'us-ascii'), (cp: 20261; name: 'x-cp20261'
), (cp: 20269; name: 'x-cp20269'), (cp: 20273; name: 'IBM273'), (cp
: 20277; name: 'IBM277'), (cp: 20278; name: 'IBM278'), (cp: 20280
; name: 'IBM280'), (cp: 20284; name: 'IBM284'), (cp: 20285; name:
'IBM285'), (cp: 20290; name: 'IBM290'), (cp: 20297; name: 'IBM297'
), (cp: 20420; name: 'IBM420'), (cp: 20423; name: 'IBM423'), (cp:
20424; name: 'IBM424'), (cp: 20833; name: 'x-EBCDIC-KoreanExtended'
), (cp: 20838; name: 'IBM-Thai'), (cp: 20866; name: 'koi8-r'), (cp
: 20871; name: 'IBM871'), (cp: 20880; name: 'IBM880'), (cp: 20905
; name: 'IBM905'), (cp: 20924; name: 'IBM00924'), (cp: 20932; name
: 'EUC-JP'), (cp: 20936; name: 'x-cp20936'), (cp: 20949; name: 'x-cp20949'
), (cp: 21025; name: 'cp1025'), (cp: 21866; name: 'koi8-u'), (cp:
28591; name: 'CP819'), (cp: 28591; name: 'IBM819'), (cp: 28591; name
: 'ISO-8859-1'), (cp: 28591; name: 'ISO-IR-100'), (cp: 28591; name
: 'ISO8859-1'), (cp: 28591; name: 'ISO_8859-1'), (cp: 28591; name
: 'ISO_8859-1:1987'), (cp: 28591; name: 'L1'), (cp: 28591; name: 'LATIN1'
), (cp: 28591; name: 'CSISOLATIN1'), (cp: 28591; name: 'iso-8859-1'
), (cp: 28591; name: 'iso8859-1'), (cp: 28592; name: 'iso-8859-2'
), (cp: 28592; name: 'iso8859-2'), (cp: 28593; name: 'iso-8859-3'
), (cp: 28593; name: 'iso8859-3'), (cp: 28594; name: 'iso-8859-4'

```

```

), (cp: 28594; name: 'iso8859-4'), (cp: 28595; name: 'iso-8859-5'
), (cp: 28595; name: 'iso8859-5'), (cp: 28596; name: 'iso-8859-6'
), (cp: 28596; name: 'iso8859-6'), (cp: 28597; name: 'iso-8859-7'
), (cp: 28597; name: 'iso8859-7'), (cp: 28598; name: 'iso-8859-8'
), (cp: 28598; name: 'iso8859-8'), (cp: 28599; name: 'iso-8859-9'
), (cp: 28599; name: 'iso8859-9'), (cp: 28603; name: 'iso-8859-13'
), (cp: 28603; name: 'iso8859-13'), (cp: 28605; name: 'iso-8859-15'
), (cp: 28605; name: 'iso8859-15'), (cp: 29001; name: 'x-Europa')
, (cp: 38598; name: 'iso-8859-8-i'), (cp: 38598; name: 'iso8859-8-i'
), (cp: 50220; name: 'iso-2022-jp'), (cp: 50221; name: 'ISO-2022-JP'
), (cp: 50221; name: 'CP50221'), (cp: 50221; name: 'ISO-2022-JP-MS'
), (cp: 50221; name: 'ISO2022-JP'), (cp: 50221; name: 'ISO2022-JP-MS'
), (cp: 50221; name: 'MS50221'), (cp: 50221; name: 'WINDOWS-50221'
), (cp: 50221; name: 'csISO2022JP'), (cp: 50222; name: 'iso-2022-jp'
), (cp: 50225; name: 'iso-2022-kr'), (cp: 50225; name: 'iso2022-kr'
), (cp: 50227; name: 'x-cp50227'), (cp: 51932; name: 'EUC-JP'), (cp
: 51932; name: 'CP51932'), (cp: 51932; name: 'MS51932'), (cp: 51932
; name: 'WINDOWS-51932'), (cp: 51932; name: 'euc-jp'), (cp: 51936
; name: 'EUC-CN'), (cp: 51949; name: 'euc-kr'), (cp: 52936; name:
'hz-gb-2312'), (cp: 54936; name: 'GB18030'), (cp: 57002; name: 'x-iscii-de'
), (cp: 57003; name: 'x-iscii-be'), (cp: 57004; name: 'x-iscii-ta'
), (cp: 57005; name: 'x-iscii-te'), (cp: 57006; name: 'x-iscii-as'
), (cp: 57007; name: 'x-iscii-or'), (cp: 57008; name: 'x-iscii-ka'
), (cp: 57009; name: 'x-iscii-ma'), (cp: 57010; name: 'x-iscii-gu'
), (cp: 57011; name: 'x-iscii-pa'), (cp: 65001; name: 'UTF-8'), (cp
: 65001; name: 'CP65001'), (cp: 65001; name: 'UTF8'))

```

UnixCpMap is a fixed structure with codepage number/codepage name pairs. It is used in [GetCodepageData \(2228\)](#), [GetSystemCodepage \(2228\)](#) and [GetCodepageByName \(2227\)](#) to map code page names to numbers and vice versa.

The map is ordered on code page number, and for equal code page numbers, the names are ordered so the most common one is used first.

UnixCpMapLimit = 406 - 83

Number of code pages in map UnixCpMap.

82.3.2 Types

82.4 Procedures and functions

82.4.1 GetCodepageByName

Synopsis: Find code page by name.

Declaration: `function GetCodepageByName(cpname: RawByteString) : TSystemCodePage`

Visibility: default

Description: `GetCodepageByName` returns the code page number matching `cpname`. The supported code page names are the ones commonly in use in `libiconv`. Names are searched case-sensitively, with the exception that `'cpN'` is converted to `'CPN'`, where N is a digit.

Errors: If no matching code page name is found, `CP_NONE` is returned.

See also: [UnixCpMap \(2227\)](#), [GetSystemCodepage \(2228\)](#), [GetCodepageData \(2228\)](#)

82.4.2 GetCodepageData

Synopsis: Return index of codepage.

Declaration: `function GetCodepageData(cp: TSystemCodePage) : LongInt`

Visibility: default

Description: `GetCodepageData` returns the index of the first entry in `UnixCpMap` (2227) which matches `cp`. Since the entries are ordered by code page number, this means the entries can be scanned for alternate code names starting at this index.

Errors: If no matching code page is found, -1 is returned.

See also: `UnixCpMap` (2227), `GetSystemCodepage` (2228), `GetCodepageByName` (2227)

82.4.3 GetSystemCodepage

Synopsis: Return the system code page based on the program environment.

Declaration: `function GetSystemCodepage : TSystemCodePage`

Visibility: default

Description: `GetSystemCodepage` returns the system code page, based on one of the environment variables `LC_ALL`, `LC_CTYPE` or `LANG`. The first non-empty variable (in the order mentioned here) is used.

Errors: If none is found, then a system default is used: Linux and Darwin use `CP_UTF8`, others use `CP_ASCII`.

See also: `UnixCpMap` (2227), `GetSystemCodepage` (2228), `GetCodepageByName` (2227)

82.5 TUnixCpData

```
TUnixCpData = record
  cp : Word;
  name : ansistring;
end
```

`TUnixCpData` contains 2 fields necessary to construct a map between code page number (`cp`) and name (`name`).

Chapter 83

Reference for unit 'unixtype'

83.1 Used units

Table 83.1: Used units by unit 'unixtype'

Name	Page
System	1335

83.2 Overview

The `unixtype` unit contains the definitions of basic UNIX types. It was initially implemented by Marco van de Voort.

When porting to a new UNIX platform, this unit should be adapted to the sizes and conventions of the platform to which the compiler is ported.

83.3 Constants, types and variables

83.3.1 Constants

`ARG_MAX` = 131072

Max number of command-line arguments.

`NAME_MAX` = 255

Max length (in bytes) of filename.

`PATH_MAX` = 4095

Max length (in bytes) of pathname.

`Prio_PGrp` = 1

#rtl.baseunix.fpGetPriority (210) option: Get process group priority.

Prio_Process = 0

#rtl.baseunix.fpGetPriority (210) option: Get process priority.

Prio_User = 2

#rtl.baseunix.fpGetPriority (210) option: Get user priority.

pthread_rwlocksize = 56

Size of pthread_rwlock_t _data structure.

SIG_MAXSIG = 128

Maximum signal number.

SYS_NMLN = 65

Max system namelength.

_PTHREAD_MUTEX_ADAPTIVE_NP = 3

Mutex options:.

_PTHREAD_MUTEX_DEFAULT = _PTHREAD_MUTEX_NORMAL

Mutex options:.

_PTHREAD_MUTEX_ERRORCHECK = _PTHREAD_MUTEX_ERRORCHECK_NP

Mutex options:.

_PTHREAD_MUTEX_ERRORCHECK_NP = 2

Mutex options: double lock returns an error code.

_PTHREAD_MUTEX_FAST_NP = _PTHREAD_MUTEX_ADAPTIVE_NP

Mutex options: Fast mutex.

_PTHREAD_MUTEX_NORMAL = _PTHREAD_MUTEX_TIMED_NP

Mutex options:.

_PTHREAD_MUTEX_RECURSIVE = _PTHREAD_MUTEX_RECURSIVE_NP

Mutex options:.

_PTHREAD_MUTEX_RECURSIVE_NP = 1

Mutex options: recursive mutex.

_PTHREAD_MUTEX_TIMED_NP = 0

Mutex options: ?

83.3.2 Types

`cbool = longbool`

Boolean type.

`cchar = cint8`

C type: 8-bit signed integer.

`cdouble = Double`

Double precision real format.

`cfloat = single`

Floating-point real format.

`cint = cint32`

C type: integer (natural size).

`cint16 = SmallInt`

C type: 16 bits sized, signed integer.

`cint32 = LongInt`

C type: 32 bits sized, signed integer.

`cint64 = Int64`

C type: 64 bits sized, signed integer.

`cint8 = ShortInt`

C type: 8 bits sized, signed integer.

`clock_t = cuint64`

Clock ticks type.

`clong = Int64`

C type: long signed integer (double sized, depending on platform 32/64 bit).

`longdouble = extended`

Usually translates to an extended, but is CPU dependent.

`longlong = cint64`

C type: 64-bit (double long) signed integer.

```
cschar = cint8
```

Signed character type.

```
cshort = cint16
```

C type: short signed integer (half sized).

```
csigned = cint
```

csigned is an alias for cint ([2231](#)).

```
csint = cint32
```

Signed integer.

```
cslong = Int64
```

The size is CPU dependent.

```
cslonglong = cint64
```

cslonglong is an alias for clonglong ([2232](#)).

```
csshort = cint16
```

Short signed integer type.

```
cuchar = cuint8
```

C type: 8-bit unsigned integer.

```
cuint = cuint32
```

C type: unsigned integer (natural size).

```
cuint16 = Word
```

C type: 16 bits sized, unsigned integer.

```
cuint32 = LongWord
```

C type: 32 bits sized, unsigned integer.

```
cuint64 = QWord
```

C type: 64 bits sized, unsigned integer.

```
cuint8 = Byte
```

C type: 8 bits sized, unsigned integer.

```
culong = QWord
```

C type: long unsigned integer (double sized, depending on platform 32/64 bit).

```
culonglong = cuint64
```

C type: 64-bit (double long) unsigned integer.

```
cunsigned = cuint
```

Alias for #rtl.unixtype.cuint ([2232](#)).

```
cushort = cuint16
```

C type: short unsigned integer (half sized).

```
dev_t = cuint64
```

Device descriptor type.

```
gid_t = cuint32
```

Group ID type.

```
ino64_t = cuint64
```

ino64_t is an inode type capable of containing 64-bit inodes.

```
ino_t = clong
```

Inode type.

```
ipc_pid_t = cint
```

Process ID.

```
kDev_t = cushort
```

Kernel device type.

```
mbstate_value_t = record
case Byte of
0: (
  __wch : wint_t;
);
1
  : (
  __wchb : Array[0..3] of Char;
);
end
```

This type should never be used directly. It is part of the `mbstate_t` (2241) type.

`mode_t = cint`

Inode mode type.

`nlink_t = cuint32`

Number of links type.

`off64_t = cint64`

64-bit offset type.

`off_t = cint64`

Offset type.

`pbool = ^cbool`

Pointer to boolean type `cbool` (2231)

`pcchar = ^cchar`

Pointer to `#rtl.UnixType.cchar` (2231).

`pcdouble = ^cdouble`

Pointer to `cdouble` (2231) type.

`pcfloat = ^cfloat`

Pointer to `cfloat` (2231) type.

`pcint = ^cint`

Pointer to `cInt` (2231) type.

`pcint16 = ^cint16`

Pointer to 16-bit signed integer type.

`pcint32 = ^cint32`

Pointer to signed 32-bit integer type.

`pcint64 = ^cint64`

Pointer to signed 64-bit integer type.

`pcint8 = ^cint8`

Pointer to 8-bits signed integer type.

```
pClock = ^clock_t
```

Pointer to TClock (2239) type.

```
pclong = ^clong
```

Pointer to cLong (2231) type.

```
pclongdouble = ^clongdouble
```

Pointer to the long double type clongdouble (2231)

```
pclonglong = ^clonglong
```

Pointer to longlong type.

```
pcschar = ^cschar
```

Pointer to character type cschar (2232).

```
pcshort = ^cshort
```

Pointer to cShort (2232) type.

```
pcsigned = ^csigned
```

Pointer to signed integer type csigned (2232).

```
pcsint = ^csint
```

Pointer to signed integer type csint (2232)

```
pcslong = ^cslong
```

Pointer to the signed long cslong (2232)

```
pcslonglong = ^cslonglong
```

Pointer to Signed longlong type cslonglong (2232)

```
pcsshort = ^csshort
```

Pointer to short signed integer type csshort (2232)

```
pcuchar = ^cuchar
```

Pointer to #rtl.UnixType.cuchar (2232).

```
pcuint = ^cuint
```

Pointer to `cUInt` (2232) type.

```
pcuint16 = ^cuint16
```

Pointer to 16-bit unsigned integer type.

```
pcuint32 = ^cuint32
```

Pointer to unsigned 32-bit integer type.

```
pcuint64 = ^cuint64
```

Pointer to unsigned 64-bit integer type.

```
pcuint8 = ^cuint8
```

Pointer to 8-bits unsigned integer type.

```
pculong = ^culong
```

Pointer to `cuLong` (2233) type.

```
pculonglong = ^culonglong
```

Unsigned longlong type.

```
pcunsigned = ^cunsigned
```

Pointer to `#rtl.unixtype.cunsigned` (2233).

```
pcushort = ^cushort
```

Pointer to `cuShort` (2233) type.

```
pDev = ^dev_t
```

Pointer to `TDev` (2239) type.

```
pGid = ^gid_t
```

Pointer to `TGid` (2239) type.

```
pid_t = cint
```

Process ID type.

```
pIno = ^ino_t
```

Pointer to `TIno` (2239) type.

```
pIno64 = ^ino64_t
```

Pointer to `ino64_t` (2233).

```
pkDev = ^kDev_t
```

Pointer to `TkDev` (2240) type.

```
pmbstate_t = ^mbstate_t
```

Pointer to `mbstate_t` (2241) type.

```
pMode = ^mode_t
```

Pointer to `TMode` (2240) type.

```
pnLink = ^nlink_t
```

Pointer to `TnLink` (2240) type.

```
pOff = ^off_t
```

Pointer to `TOff` (2240) type.

```
pOff64 = ^off64_t
```

Pointer to `off64_t` type.

```
pPid = ^pid_t
```

Pointer to `TPid` (2240) type.

```
pSize = ^size_t
```

Pointer to `TSize` (2240) type.

```
psize_t = pSize
```

Pointer to `size_t` (2239) type.

```
pSockLen = ^socklen_t
```

Pointer to `TSockLen` (2240) type.

```
pSSize = ^ssize_t
```

Pointer to `TsSize` (2240) type.

```
PStatFS = ^TStatfs
```

Pointer to `TStatFS` (2243) type.

```
pthread_key_t = cuint
```

Thread local storage key (opaque).

```
PTHREAD_MUTEX_T = record
case Byte of
1: (
  __m_reserved : LongInt
  ;
  __m_count : LongInt;
  __m_owner : pointer;
  __m_kind : LongInt
  ;
  __m_lock : record
    __status : SizeInt;
    __spinlock : LongInt
  ;
  end;
);
end
```

`_pthread_mutex_t` describes a thread mutex. It should be considered an opaque record, the names of the fields can change anytime.

```
pthread_rwlock_t = record
case Boolean of
False: (
  _data : Array
    [0..pthread_rwlocksize-1] of Char;
);
True: (
  align : clong;
);
end
```

`pthread_rwlock_t` describes a lock. It should be considered an opaque record, the names of the fields can change anytime.

```
pthread_t = culong
```

Thread description record.

```
pTime = ^time_t
```

Pointer to `TTime` (2240) type.

```
ptimespec = ^timespec
```

Pointer to `timespec` (2243) record.

```
ptimeval = ^timeval
```

Pointer to `timeval` (2243) record.

`ptime_t = ^time_t`

Pointer to `time_t` (2239) type.

`pUId = ^uid_t`

Pointer to `TUId` (2240) type.

`pwchar_t = ^wchar_t`

Pointer to `wchar_t` (2241) type.

`size_t = uint64`

Size specification type.

`socklen_t = uint32`

Socket address length type.

`ssize_t = int64`

Small size type.

`TClock = clock_t`

Alias for `clock_t` (2231) type.

`TDev = dev_t`

Alias for `dev_t` (2233) type.

`TGid = gid_t`

Alias for `gid_t` (2233) type.

`time_t = int64`

Time span type.

`TIno = ino_t`

Alias for `ino_t` (2233) type.

`TIno64 = ino64_t`

Alias for `ino64_t` (2233).

`TIOCtlRequest = culong`

Opaque type used in `FpIOCtl` (211).

TkDev = kDev_t

Alias for kDev_t (2233) type.

TMode = mode_t

Alias for mode_t (2234) type.

TnLink = nlink_t

Alias for nlink_t (2234) type.

TOff = off_t

Alias for off_t (2234) type.

TOff64 = off64_t

Alias for off64_t type.

TPid = pid_t

Alias for pid_t (2236) type.

TSize = size_t

Alias for size_t (2239) type.

TSockLen = socklen_t

Alias for socklen_t (2239) type.

TSSize = ssize_t

Alias for ssize_t (2239) type.

TTime = time_t

Alias for TTime (2240) type.

TTimeSpec = timespec

Alias for TimeSpec (2243) type.

TTimeVal = timeval

Alias for TimeVal (2243) record.

TUid = uid_t

Alias for uid_t (2241) type.

```
uid_t = cuint32
```

User ID type.

```
wchar_t = cint32
```

Wide character type.

```
wint_t = cint32
```

Wide character size type.

83.4 mbstate_t

```
mbstate_t = record
  __count : cint;
  __value : mbstate_value_t;
end
```

This type should never be used directly.

83.5 pthread_attr_t

```
pthread_attr_t = record
  __detachstate : cint;
  __schedpolicy :
  cint;
  __schedparam : sched_param;
  __inheritsched : cint;
  __scope
  : cint;
  __guardsize : size_t;
  __stackaddr_set : cint;
  __stackaddr
  : pointer;
  __stacksize : size_t;
end
```

`pthread_attr_t` describes the thread attributes. It should be considered an opaque record, the names of the fields can change anytime. Use the appropriate functions to set the thread attributes.

83.6 pthread_condattr_t

```
pthread_condattr_t = record
  __dummy : cint;
end
```

`pthread_condattr_t` describes the attributes of a thread mutex. It should be considered an opaque record, the names of the fields can change anytime.

83.7 pthread_cond_t

```
pthread_cond_t = record
  __c_lock : _pthread_fastlock;
  __c_waiting
  : pointer;
  __padding : Array[0..48-1-sizeof(_pthread_fastlock)
    -sizeof(pointer)-sizeof(clonglong)] of Byte;
  __align : clonglong
;
end
```

pthread_cond_t describes a thread conditional variable. It should be considered an opaque record, the names of the fields can change anytime.

83.8 pthread_mutexattr_t

```
pthread_mutexattr_t = record
  __mutexkind : cint;
end
```

pthread_mutexattr_t describes the attributes of a thread mutex. It should be considered an opaque record, the names of the fields can change anytime.

83.9 pthread_rwlockattr_t

```
pthread_rwlockattr_t = record
  __lockkind : cint;
  __pshared : cint
;
end
```

pthread_rwlockattr_t describes the attributes of a lock. It should be considered an opaque record, the names of the fields can change anytime.

83.10 sched_param

```
sched_param = record
  __sched_priority : cint;
end
```

Scheduling parameter description record.

83.11 sem_t

```
sem_t = record
```

```

__sem_lock : _pthread_fastlock;
__sem_value : cint
;
__sem_waiting : pointer;
end

```

`sem_t` describes a thread semaphore. It should be considered an opaque record, the names of the fields can change anytime.

83.12 timespec

```

timespec = record
  tv_sec : time_t;
  tv_nsec : clong;
end

```

Record specifying time interval.

83.13 timeval

```

timeval = record
  tv_sec : time_t;
  tv_usec : clong;
end

```

Time specification type.

83.14 TStatfs

```

TStatfs = record
  fstype : clong;
  bsize : clong;
  blocks : culong
;
  bfree : culong;
  bavail : culong;
  files : culong;
  ffree
  : culong;
  fsid : Array[0..1] of cint;
  namelen : clong;
  frsize
  : clong;
  flags : clong;
  spare : Array[0..3] of clong;
end

```

Record describing a file system in the `unix.fstatfs` ([2229](#)) call.

83.15 `_pthread_fastlock`

```
_pthread_fastlock = record
  __status : clong;
  __spinlock : cint
;
end
```

`_pthread_fastlock` describes a thread mutex. It should be considered an opaque record, the names of the fields can change anytime.

Chapter 84

Reference for unit 'unixutil'

84.1 Used units

Table 84.1: Used units by unit 'unixutil'

Name	Page
BaseUnix	155
System	1335

84.2 Overview

The UnixUtil unit contains some of the routines that were present in the old Linux unit, but which do not really belong in the UNIX ([2189](#)) or baseunix ([155](#)) units.

Most of the functions described here have cross-platform counterparts in the SysUtils ([1609](#)) unit. It is therefore recommended to use that unit.

84.3 Procedures and functions

84.3.1 ArrayStringToPPchar

Synopsis: Convert an array of string to an array of null-terminated strings.

Declaration: `function ArrayStringToPPchar(const S: Array of RawByteString;
reserveentries: LongInt) : PPChar`

Visibility: default

Description: `ArrayStringToPPchar` creates an array of null-terminated strings that point to strings which are the same as the strings in the array `S`. The function returns a pointer to this array. The array and the strings it contains must be disposed of after being used, because it they are allocated on the heap.

The `ReserveEntries` parameter tells `ArrayStringToPPchar` to allocate room at the end of the array for another `ReserveEntries` entries.

Errors: If not enough memory is available, an error may occur.

See also: `StringToPPChar` ([2246](#))

84.3.2 StringToPPChar

Synopsis: Split string in list of null-terminated strings.

Declaration: `function StringToPPChar(S: PChar; ReserveEntries: Integer) : PPChar`
`function StringToPPChar(var S: RawByteString; ReserveEntries: Integer)`
`: PPChar`

Visibility: default

Description: `StringToPPChar` splits the string `S` in words, replacing any whitespace with zero characters. It returns a pointer to an array of pchars that point to the first letters of the words in `S`. This array is terminated by a `Nil` pointer.

The function does *not* add a zero character to the end of the string unless it ends on whitespace.

The function reserves memory on the heap to store the array of `PChar`; The caller is responsible for freeing this memory.

This function can be called to create arguments for the various `Exec` calls.

Errors: None.

See also: `ArrayStringToPPChar` (2245), `#rtl.baseunix.FpExecve` (201)

Listing: `./examples/unutilex/ex70.pp`

Program Example70;

{ Program to demonstrate the StringToPPchar function. }

Uses UnixUtil;

Var `S : String;`
`P : PPChar;`
`I : longint;`

begin
// remark whitespace at end.
`S:='This is a string with words. ';`
`P:=StringToPPChar(S,0);`
`I:=0;`
While `P[I]<>Nil` **do**
 begin
 Writeln('Word ',I,' : ',P[I]);
 Inc(I);
 end;
FreeMem(P,I***SizeOf**(Pchar));
end.

Chapter 85

Reference for unit 'Variants'

85.1 Used units

Table 85.1: Used units by unit 'Variants'

Name	Page
rtlconsts	??
sysconst	??
System	1335
sysutils	1609
TypeInfo	2059

85.2 Overview

The compiler has built-in support for variants, and for many operations, variants can be used without thinking about it. The system unit has built-in support for some of the basic operations on a variant, as well as some compiler helper routines. However, some operations and definitions are implemented in the `Variants` unit so as not to burden the system unit with routines that may not always be needed.

There is a basic set of variants that are defined by the Windows OS, these are supported by the compiler and the routines in the `Variants` unit. Additional variant types can be registered using the `TCustomVariantType` ([2276](#)) type.

The variants unit also registers a handler for setting published properties (using RTTI) using variant-typed values. Nothing needs to be done for this except including the variants unit in your program.

85.3 Constants, types and variables

85.3.1 Constants

`CFirstUserType = CMinVarType + CIncVarType`

`CFirstUserType` is the first allocated value for `vType` when registering a variant type by the RTL, when instantiating a `TCustomVariantType` ([2276](#)) to register a variant type. It is better not to specify a custom variant type value, but let the system allocate a custom variant type.


```
CIncVarType = $000F
```

`CIncVarType` specifies the width of the gap after `CMinVarType` (2248). No user types are registered between `CMinVarType` and `CMinVarType+CIncVarType`.

```
CMaxNumberOfCustomVarTypes = $0EFF
```

`CMaxNumberOfCustomVarTypes` is the amount of custom variant types that can be registered. The range of the variant type indicator (`vType`) only allows a limited amount of variants. This constant is the maximum amount.

```
CMaxVarType = CMinVarType + CMaxNumberOfCustomVarTypes
```

`CMaxVarType` is the maximum allowed value for `vType` when registering a variant type, instantiating a `TCustomVariantType` (2276) to register a variant type. It is better not to specify a value, but let the system allocate a custom variant type.

```
CMinVarType = $0100
```

`CMinVarType` is the first allowed value for `vType` when registering a variant type by the OS, instantiating a `TCustomVariantType` (2276) to register a variant type. It is better not to specify a custom variant type value, but let the system allocate a custom variant type.

```
FloatVarTypes = [varSingle, varDouble, varCurrency, varDecimal]
```

`FloatlVarTypes` is used in `VarIsFloat` (2265) to decide which variant types are considered ordinals.

```
OrdinalVarTypes = [varSmallInt, varInteger, varBoolean, varShortInt,
    , varByte, varWord, varLongWord, varInt64, varQWord]
```

`OrdinalVarTypes` is used in `VarIsOrdinal` (2265) to decide which variant types are considered ordinals.

```
VarOpAsText : Array[TVarOp] of AnsiString = ('+', '-', '*', '/', 'div',
    , 'mod', 'shl', 'shr', 'and', 'or', 'xor', '', '-', 'not', '=', '<>',
    , '<', '<=', '>', '>=', '**')
```

`VarOpAsText` is an array with the names of the various variant operations.

85.3.2 Types

```
TAnyProc = procedure(var V: tvardata)
```

`TAnyProc` is the type for the `ClearAnyProc` (2250), `ChangeAnyProc` (2250) and `RefAnyProx` (2247) callbacks. It accepts a reference to a variant record. The operation to be performed depends on the callback.

```
TBooleanToStringRule = (bsrAsIs, bsrLower, bsrUpper)
```

Table 85.2: Enumeration values for type TBooleanToStringRule

Value	Explanation
bsrAsIs	Leave casing as is.
bsrLower	Convert to lowercase.
bsrUpper	Convert to uppercase.

TBooleanToStringRule describes how boolean values are converted to string values. It is defined for Delphi compatibility, but is not used in the FPC runtime.

TCustomVariantTypeClass = Class of TCustomVariantType

TCustomVariantTypeClass is the class type of TCustomVariantType.

TNullCompareRule = (ncrError, ncrStrict, ncrLoose)

Table 85.3: Enumeration values for type TNullCompareRule

Value	Explanation
ncrError	Raise an error when one of the values is Null.
ncrLoose	Attempt to compare anyway.
ncrStrict	Act as if the comparison is false.

TNullCompareRule is the type for the NullEqualityRule (2251) and NullMagnitudeRule (2251) variables. It can have the following values:

ncrError Raise an error when one of the values is Null.

ncrStrict Act as if the comparison is false.

ncrLoose Attempt to compare anyway.

TVarCompareResult = (crLessThan, crEqual, crGreaterThan)

Table 85.4: Enumeration values for type TVarCompareResult

Value	Explanation
crEqual	Both values are equal.
crGreaterThan	The first value is greater than the second value.
crLessThan	The first value is less than the second value.

TVarCompareResult is used when comparing 2 custom variant values in TCustomVariantType.Compare (2280). The following values exist:

crLessThan The first value is less than the second value.

crGreaterThan The first value is greater than the second value.

crEqual Both values are equal.

```
TVarDataArray = Array of tvardata
```

TVarDataArray is a helper type for instance used in IVarInvokeable.DoFunction (2275) to represent the arguments passed to a function.

```
TVarDispProc = procedure(Dest: PVariant; const Source: Variant;
    CallDesc: pcalldesc; Params: Pointer)
```

TVarDispProc is the type for the VarDispProc (2252) callback handler. It accepts a destination variant (Dest) for a result. Source is the variant on which the operation was invoked, CallDesc Describes the arguments to the call and Params points to the parameters provided in the call.

```
TVariantRelationship = (vrEqual, vrLessThan, vrGreaterThan, vrNotEqual
    )
```

Table 85.5: Enumeration values for type TVariantRelationship

Value	Explanation
vrEqual	Are the 2 variants equal.
vrGreaterThan	Is the first variant (strictly) greater than the second.
vrLessThan	Is the first variant (strictly) less than the second.
vrNotEqual	Are the 2 variants unequal.

TVariantRelationship is used by VarCompareValue (2261) to indicate the type of comparison operation it must perform. It has the following values:

vrEqual Are the 2 variants equal.

vrLessThan Is the first variant (strictly) less than the second.

vrGreaterThan Is the first variant (strictly) greater than the second.

vrEqual Are the 2 variants equal.

vrEqual Are the 2 variants equal.

85.3.3 Variables

```
ChangeAnyProc : TAnyProc
```

ChangeAnyProc is currently not used in the Variants unit.

```
ClearAnyProc : TAnyProc
```

ClearAnyProc is called when the system needs to clear a variant of type varAny. it must clear the variant.

```
EmptyParam : OleVariant
```

`EmptyParam` is an initialized variant with type `varError` and error value `VAR_PARAMNOTFOUND`. In difference with `Null` (2254) and `UnAssigned` (2255) it is a variable. You should take care never to write to it.

```
InvalidCustomVariantType : TCustomVariantType
```

`InvalidCustomVariantType` can be set to a `TCustomVariantType` (2276) instance to indicate an unknown type. By default it is set to `Pointer(-1)`.

```
NullAsStringValue : AnsiString = ''
```

`NullAsStringValue` is the value used when converting a `Null` to a string. It is only used when `NullStrictConvert` (2251) is `False`, if `NullStrictConvert` (2251) is `True`, the value of `NullAsStringValue` is ignored.

```
NullEqualityRule : TNullCompareRule = ncrLoose
```

`NullEqualityRule` is checked when the system needs to compare the equality of variants with `Null` values (operations `opCmpEq`, `opCmpNe`). Check `TNullCompareRule` (2249) for a list of allowed values. The default is `ncrLoose`.

For determining the order (less than, greater than etc.) see `NullMagnitudeRule` (2251).

```
NullMagnitudeRule : TNullCompareRule = ncrLoose
```

`NullEqualityRule` is checked when the system needs to determine the ordering of variants with `Null` values (operations like `opCmpGe`, `opCmpLe`). Check `TNullCompareRule` (2249) for a list of allowed values. The default is `ncrLoose`.

For determining equality (`opCmpEq`, `opCmpNe`) of `Null` values, see `NullEqualityRule` (2251).

```
NullStrictConvert : Boolean = True
```

`NullStrictConvert` determines what to do when typecasting a `Null` value to another type: If it is `True` then an exception will be raised using `VarCastError` (2260). If it is `False` then a sensible default is used: 0 or some variation on 0 based on the type of the variant. For string values the `NullAsStringValue` (2251) is used.

```
OleVariantInt64AsDouble : Boolean = False
```

`OleVariantInt64AsDouble` describes what to do when a `Int64` value must be converted to a `OleVariant` value. When `True` the `Int64` variant is cast to a double. When `False`, it remains an `int64` value.

```
PackVarCreation : Boolean = True
```

`PackVarCreation` determines what to do when a variant array is created for elements with an integer type (`varSmallint`, `varByte` and the like). A value of `False` means the array will contain `varInteger` elements. This can result in better memory alignment. When `PackVarCreation` is `True` then the smallest possible size is selected for the elements.

```
RefAnyProc : TAnyProc
```

`RefAnyProc` is called when the system needs to obtain a reference to a variant of type `varAny`. it must replace the variant with a reference to the variant.

`VarDispProc` : `TVarDispProc`

`VarDispProc` is the handler invoked when a dispatch invoke is handled on a variant of type `varDispatch`, `varAny` or `varUnknown`.

85.4 Procedures and functions

85.4.1 DynArrayFromVariant

Synopsis: Convert a variant to a dynamic array.

Declaration: `procedure DynArrayFromVariant(var DynArray: Pointer; const V: Variant; TypeInfo: Pointer)`

Visibility: default

Description: `DynArrayFromVariant` transforms a variant array to a dynamic array. It uses `TypeInfo` to calculate the length, element type and dimension of the array.

The opposite transformation can be performed with `DynArrayToVariant` (2252).

See also: `VarArrayCreateError` (2256), `VarArrayCreate` (2255), `VarArrayOf` (2258), `DynArrayToVariant` (2252)

85.4.2 DynArrayToVariant

Synopsis: Convert a Dynamic Array To a Variant.

Declaration: `procedure DynArrayToVariant(var V: Variant; const DynArray: Pointer; TypeInfo: Pointer)`

Visibility: default

Description: `DynArrayToVariant` converts the dynamic array `DynArray` to a variant array `V`. It uses the type information in `TypeInfo` to calculate the number of dimensions, array lengths and type of the element. The dynamic array can only contain basic types.

If there is no data, an empty variant will be returned.

The opposite transformation can be performed with `DynArrayFromVariant` (2252).

See also: `VarArrayCreateError` (2256), `VarArrayCreate` (2255), `VarArrayOf` (2258), `DynArrayFromVariant` (2252)

85.4.3 FindCustomVariantType

Synopsis: Find a custom variant class on `vartype`.

Declaration: `function FindCustomVariantType(const aVarType: tvartype; out CustomVariantType: TCustomVariantType) : Boolean; Overload`
`function FindCustomVariantType(const TypeName: AnsiString; out CustomVariantType: TCustomVariantType) : Boolean; Overload`

Visibility: default

Description: `FindCustomVariantType` searches the registry of known `TCustomVariantType` (2276) classes and returns the instance registered for the variant type `aVarType` or the class name `TypeNames` in `CustomVariantType` if found.

It returns `True` if it found a matching definition, or `False` otherwise.

See also: `TCustomVariantType` (2276)

85.4.4 FindVarData

Synopsis: Return a pointer to variant data.

Declaration: `function FindVarData(const V: Variant) : pvardata`

Visibility: default

Description: `FindVarData` returns a pointer to the argument `V` if it is not a reference. If it is a reference, then the reference pointer is returned.

85.4.5 GetPropValue

Synopsis: Return a property value as a variant.

Declaration: `function GetPropValue(Instance: TObject; PropInfo: PPropInfo;
PreferStrings: Boolean) : Variant; Overload`

Visibility: default

Description: `GetPropValue` returns the value of the property described by `PropInfo` from the `Instance`. The property value is returned as a variant type.

when `PreferStrings` is true, the implementation will tend to cast to a string-valued variant when a conversion must be done.

This function is used as the value for the `TypeInfo` (2247) unit's `OnGetPropValue` (2247) callback. This callback is automatically initialized with the function when the variants unit is used.

See also: `TypeInfo` (2247), `TypeInfo.OnGetPropValue` (2247), `SetPropValue` (2254)

85.4.6 GetVariantProp

Synopsis: Get variant valued property.

Declaration: `function GetVariantProp(Instance: TObject; PropInfo: PPropInfo)
: Variant
function GetVariantProp(Instance: TObject; const PropName: AnsiString)
: Variant`

Visibility: default

Description: `GetVariantProp` returns the value of the variant-types property described by `PropInfo` or `PropName` from the `Instance`.

This function is used as the value for the `TypeInfo` (2247) unit's `OnGetVariantProp` (2247) callback. This callback is automatically initialized with the function when the variants unit is used.

See also: `TypeInfo` (2247), `TypeInfo.OnGetVariantProp` (2247), `SetVariantProp` (2255), `GetPropValue` (2253)

85.4.7 HandleConversionException

Synopsis: Convert an exception to a variant exception.

Declaration: `procedure HandleConversionException(const ASourceType: tvartype;
const ADestType: tvartype)`

Visibility: default

Description: `HandleConversionException` converts a RTL exception (`EConvertError` (2247) or `ERangeError` (2247) or `EOverflow` (2247)) to an appropriate variant error (`varCastError` (2260) and `varOverflowError` (2267)). Other exceptions are re-raised. If a source and destination type `ASourceType` and `ADestType` are specified they are included in the error message.

See also: `EConvertError` (2247), `ERangeError` (2247), `varCastError` (2260), `varOverflowError` (2267), `EOverflow` (2247)

85.4.8 Null

Synopsis: Return a null variant.

Declaration: `function Null : Variant`

Visibility: default

Description: `UnAssigned` returns a Null variant (`type = varNull`). It can be used to test for equality with a Null variant.

See also: `UnAssigned` (2255), `EmptyParam` (2251)

85.4.9 SetClearVarToEmptyParam

Synopsis: Create an error variant with value `VAR_PARAMNOTFOUND`.

Declaration: `procedure SetClearVarToEmptyParam(var V: tvardata)`

Visibility: default

Description: `SetClearVarToEmptyParam` clears the variant `Vvar` and sets its type to `varError` and value to `VAR_PARAMNOTFOUND`. `VarIsEmptyParam(SetClearVarToEmptyParam(V))` will return `True`.

Errors: None.

See also: `VarIsEmptyParam` (2264), `VarIsError` (2264)

85.4.10 SetPropValue

Synopsis: Set a property value as a variant.

Declaration: `procedure SetPropValue(Instance: TObject; PropInfo: PPropInfo;
const Value: Variant); Overload`

Visibility: default

Description: `SetPropValue` sets the value of the property described by `PropInfo` from the `Instance`. The property value is set from the variant-typed value `Value`.

This function is used as the value for the `TypeInfo` (2247) unit's `OnSetPropValue` (2247) callback. This callback is automatically initialized with the function when the variants unit is used.

See also: `TypeInfo` (2247), `TypeInfo.OnSetPropValue` (2247), `GetPropValue` (2253)

85.4.11 SetVariantProp

Synopsis: Set variant valued property.

Declaration:

```
procedure SetVariantProp(Instance: TObject; const PropName: AnsiString;
                        const Value: Variant)
procedure SetVariantProp(Instance: TObject; PropInfo: PPropInfo;
                        const Value: Variant)
```

Visibility: default

Description: `SetVariantProp` sets the value of the variant-types property described by `PropInfo` or `PropName` from the `Instance` to `Value`.

This function is used as the value for the `TypeInfo` (2247) unit's `OnSetVariantProp` (2247) callback. This callback is automatically initialized with the function when the variants unit is used.

See also: `TypeInfo` (2247), `TypeInfo.OnSetVariantProp` (2247), `GetVariantProp` (2253), `GetPropValue` (2253)

85.4.12 Unassigned

Synopsis: Return an unassigned variant.

Declaration:

```
function Unassigned : Variant
```

Visibility: default

Description: `UnAssigned` returns an unassigned variant (`type = varEmpty`). It can be used to test for equality with an empty variant.

See also: `Null` (2254), `EmptyParam` (2251)

85.4.13 VarArrayAsPSafeArray

Synopsis: Return internal array of variant value.

Declaration:

```
function VarArrayAsPSafeArray(const A: Variant) : pvararray
```

Visibility: default

Description: `VarArrayAsPSafeArray` returns the internal array of the variant `A` if it is a variant array. If not, an exception is raised.

Errors: if the variant `A` is not an array, an `EVariantInvalidArgError` (2272) exception is raised.

See also: `VarIsArray` (2263), `EVariantInvalidArgError` (2272)

85.4.14 VarArrayCreate

Synopsis: Create a variant array.

Declaration:

```
function VarArrayCreate(const Bounds: Array of SizeInt;
                        aVarType: tvartype) : Variant
function VarArrayCreate(const Bounds: pvararrayboundarray;
                        Dims: SizeInt; aVarType: tvartype) : Variant
```

Visibility: default

Description: `VarArrayCreate` creates a (optionally multidimensional) array with upper,lower bounds specified in `Bounds`. The number of bounds (in case of a single array) must be even: 2 bounds for every dimension of the array are required. All elements of the array are of the same type. The following examples create a one-dimensional array with 10 elements

```
VarArrayCreate([0,9],varInteger);
VarArrayCreate([1,10],varInteger);
```

The first array is 0-based, the second is 1-based. The following creates a 2-dimensional array:

```
VarArrayCreate([0,9,0,1],varInteger);
VarArrayCreate([1,10,1,2],varInteger);
```

The first array is 0-based, the second is 1-based. Each array consists of an array of 2 elements.

The array can also be specified as a pointer to array of `system.tvararraybound` (2247) records, and a number of dimensions. The above 1-dimensional arrays can be specified as:

```
var
  B : tvararraybound;
begin
  b.elementcount:=10;
  B.lowbound:=0;
  VarArrayCreate(@B,1,varInteger);
  b.elementcount:=10;
  B.lowbound:=1;
  VarArrayCreate(@B,1,varInteger);
```

Errors: If an uneven amount of bounds is specified or the operating system failed to create the array, an exception is raised using `VarArrayCreateError` (2256)

See also: `VarArrayCreateError` (2256), `VarArrayOf` (2258)

85.4.15 VarArrayCreateError

Synopsis: Raise an `EVariantArrayCreateError` error.

Declaration: `procedure VarArrayCreateError`

Visibility: default

Description: `VarArrayCreateError` raises an `EVariantArrayCreateError` (2271) exception with a standard error message.

See also: `EVariantArrayCreateError` (2271), `VarOverflowError` (2267), `VarInvalidNullOp` (2262), `VarInvalidOp` (2262), `VarBadIndexError` (2259), `VarArrayLockedError` (2257)

85.4.16 VarArrayDimCount

Synopsis: Return the number of dimensions of the array.

Declaration: `function VarArrayDimCount(const A: Variant) : LongInt`

Visibility: default

Description: `VarArrayDimCount` returns the number of dimensions of the array `A`. If `A` is not an array, zero is returned.

Errors: None.

See also: `VarArrayCreate` ([2255](#)), `VarArrayLowBound` ([2258](#)), `VarArrayHighBound` ([2257](#))

85.4.17 `VarArrayHighBound`

Synopsis: Return lower bound of an array.

Declaration: `function VarArrayHighBound(const A: Variant; Dim: LongInt) : LongInt`

Visibility: default

Description: `VarArrayHighBound` returns the high bound (max index) of dimension `Dim` of array `A`. The dimension `Dim` is 1-based.

Errors: If the dimension is out of range, or `A` is not a variant array, an exception will be raised.

See also: `VarArrayCreate` ([2255](#)), `VarArrayDimCount` ([2256](#)), `VarArrayLowBound` ([2258](#))

85.4.18 `VarArrayLock`

Synopsis: get a pointer to data of a variant array.

Declaration: `function VarArrayLock(const A: Variant) : Pointer`

Visibility: default

Description: `VarArrayLock` returns a pointer to the data of an array-typed variant `A` and locks the variant. The pointer can then be used to manipulate the data of the array and be sure that the data is not modified by another process or thread. While the data is not released using `VarArrayUnlock` ([2258](#)) the variant cannot be changed or released, it is therefor imperative that the variant is again released. Memory leaks will be the result if this is not done.

See also: `VarArrayUnlock` ([2258](#))

85.4.19 `VarArrayLockedError`

Synopsis: Raise an `EVariantArrayLockedError` error.

Declaration: `procedure VarArrayLockedError`

Visibility: default

Description: `VarArrayLockedError` raises an `EVariantArrayLockedError` ([2271](#)) exception with a standard error message.

See also: `EVariantArrayLockedError` ([2271](#)), `VarInvalidNullOp` ([2262](#)), `VarInvalidOp` ([2262](#)), `VarBadIndexError` ([2259](#))

85.4.20 VarArrayLowBound

Synopsis: Return lower bound of an array.

Declaration: `function VarArrayLowBound(const A: Variant; Dim: LongInt) : LongInt`

Visibility: default

Description: `VarArrayLowBound` returns the lower bound (min index) of dimension `Dim` of array `A`. The dimension `Dim` is 1-based.

Errors: If the dimension is out of range, or `A` is not a variant array, an exception will be raised.

See also: `VarArrayCreate` (2255), `VarArrayDimCount` (2256), `VarArrayHighBound` (2257)

85.4.21 VarArrayOf

Synopsis: Create a variants array of a series of values.

Declaration: `function VarArrayOf(const Values: Array of Variant) : Variant`

Visibility: default

Description: `VarArrayOf` creates a variant array with elements of type `varVariant`. The array has as many values as there are elements in `Values` and the element values are copied from `Values`.

Errors: If the array cannot be created, an `EVariantError` exception may be raised.

See also: `VarArrayCreateError` (2256), `VarArrayCreate` (2255), `DynArrayToVariant` (2252)

85.4.22 VarArrayRef

Synopsis: Get a reference to a variant array.

Declaration: `function VarArrayRef(const A: Variant) : Variant`

Visibility: default

Description: `VarArrayRef` returns a reference to the variant array `A`.

Errors: An exception will be raised if the variant `A` is not an array.

See also: `VarArrayLock` (2257), `VarArrayUnlock` (2258), `VarIsArray` (2263)

85.4.23 VarArrayUnlock

Synopsis: Release data captured by `VarArrayLock`.

Declaration: `procedure VarArrayUnlock(const A: Variant)`

Visibility: default

Description: `VarArrayUnlock` unlocks the variant array `A` that was previously locked using `VarArrayLock` (2257).

See also: `VarArrayLock` (2257)

85.4.24 VarAsError

Synopsis: Create an error-typed variant.

Declaration: `function VarAsError (AResult: HRESULT) : Variant`

Visibility: default

Description: `VarAsError` creates an error-typed variant with value `aResult`.

Errors: None.

See also: `SetClearVarToEmptyParam` (2254), `VarIsError` (2264)

85.4.25 VarAsType

Synopsis: Attempt to cast a variant to another type.

Declaration: `function VarAsType (const V: Variant; aVarType: tvartype) : Variant`

Visibility: default

Description: `VarAsType` attempts to cast the variant `V` to a new variant of type `aVarType` and returns the new variant.

Errors: If the requested type cast is not supported or possible, an `EVariantTypeCastError` (2274) exception may be raised.

See also: `EVariantTypeCastError` (2274), `VarIsType` (2266), `VarAsType` (2259)

85.4.26 VarBadIndexError

Synopsis: Raise an `EVariantBadIndexError` error.

Declaration: `procedure VarBadIndexError`

Visibility: default

Description: `VarBadIndexError` raises an `EVariantBadIndexError` (2272) exception with a standard error message.

See also: `EVariantBadIndexError` (2272), `VarInvalidNullOp` (2262), `VarInvalidOp` (2262), `VarArrayLockedError` (2257)

85.4.27 VarBadTypeError

Synopsis: Raise an `EVariantBadVarTypeError` error.

Declaration: `procedure VarBadTypeError`

Visibility: default

Description: `VarBadTypeError` raises an `EVariantBadVarTypeError` (2272) exception.

See also: `EVariantBadVarTypeError` (2272), `VarInvalidNullOp` (2262), `VarInvalidOp` (2262)

85.4.28 VarCastError

Synopsis: Raise a variant type cast error `EVariantTypeCastError`.

Declaration: `procedure VarCastError`
`procedure VarCastError(const ASourceType: tvartype;`
`const ADestType: tvartype)`

Visibility: default

Description: `VarCastError` raises an `EVariantTypeCastError` (2274) exception. If the source and destination types are specified, then a description of the types is included in the error message.

See also: `EVariantTypeCastError` (2274), `VarCastErrorOle` (2260)

85.4.29 VarCastErrorOle

Synopsis: Raise a variant type cast error `EVariantTypeCastError`.

Declaration: `procedure VarCastErrorOle(const ASourceType: tvartype)`

Visibility: default

Description: `VarCastError` raises an `EVariantTypeCastError` (2274) exception. a description of the source type (`aSourceType`) is included in the error message.

See also: `EVariantTypeCastError` (2274), `VarCastError` (2260)

85.4.30 VarCheckEmpty

Synopsis: Raise exception if a variant is empty.

Declaration: `procedure VarCheckEmpty(const V: Variant)`

Visibility: default

Description: `VarCheckEmpty` will raise an `EVariantError` (2247) exception if the variant `V` is empty (it uses `VarIsEmpty` (2264) to check this.)

Errors: if the variant is empty an `EVariantError` (2247) exception is raised.

See also: `VarIsEmpty` (2264), `VarType` (2270), `VarIsNull` (2265)

85.4.31 VarClear

Synopsis: Clear the variant value.

Declaration: `procedure VarClear(var V: Variant)`
`procedure VarClear(var V: OleVariant)`

Visibility: default

Description: `VarClear` clears the variant, possibly freeing any memory taken by the value

Errors: None.

See also: `SetClearVarToEmptyParam` (2254), `VarIsError` (2264), `VarIsClear` (2263)

85.4.32 VarCompareValue

Synopsis: Compare 2 variant values.

Declaration: `function VarCompareValue(const A: Variant; const B: Variant)
: TVariantRelationship`

Visibility: default

Description: `VarCompareValue` compares 2 variants A and B. It returns one of the following values:

vrEqual if the 2 variant values are equal.

vrUnequal if one of the 2 variant is null or empty and the other is not.

vrGreaterThan if A>B

vrLessThan if A<B

See also: `VarSameValue` ([2267](#))

85.4.33 VarCopyNoInd

Synopsis: Not supported.

Declaration: `procedure VarCopyNoInd(var Dest: Variant; const Source: Variant)`

Visibility: default

Description: `VarCopyNoInd` is currently not supported.

Errors: An `EVariantError` exception is always raised.

85.4.34 VarEnsureRange

Synopsis: Make sure the variant is within a specified range.

Declaration: `function VarEnsureRange(const AValue: Variant; const AMin: Variant;
const AMax: Variant) : Variant`

Visibility: default

Description: `VarEnsureRange` checks `AValue` and returns it if it is in the range specified by `AMin`, `AMax`. If it is less than `AMin`, then `AMin` is returned. If it is larger than `AMax`, then `AMax` is returned.

Errors: If the variants are of different types and they cannot be converted, an exception will be raised.

See also: `VarInRange` ([2262](#))

85.4.35 VarFromDateTime

Synopsis: Create variant from `TDateTime` value.

Declaration: `function VarFromDateTime(const DateTime: TDateTime) : Variant`

Visibility: default

Description: `VarFromDateTime` creates a variant with type `varDate` and value `DateTime`.

Errors: None.

See also: `varToDateTime` ([2268](#))

85.4.36 VarInRange

Synopsis: Check if a variant is in a range of values.

Declaration: `function VarInRange(const AValue: Variant; const AMin: Variant;
const AMax: Variant) : Boolean`

Visibility: default

Description: `VarInRange` is an auxiliary function which checks whether `AValue` is in the range defined by `AMin` and `AMax`, borders included.

Errors: If the variants are of different types and they cannot be converted, an exception will be raised.

See also: `VarEnsureRange` ([2261](#))

85.4.37 VarInvalidArgError

Synopsis: Raise an `EVariantInvalidArgError` error.

Declaration: `procedure VarInvalidArgError
procedure VarInvalidArgError(AType: tvartype)`

Visibility: default

Description: `VarInvalidArgError` raises an `EVariantInvalidArgError` ([2272](#)) exception with a standard error message. If the `AType` argument is specified, a description of the argument type is included in the error message.

See also: `EVariantInvalidArgError` ([2272](#)), `VarInvalidNullOp` ([2262](#)), `VarInvalidOp` ([2262](#)), `VarBadIndexError` ([2259](#)), `VarArrayLockedError` ([2257](#))

85.4.38 VarInvalidNullOp

Synopsis: Raise an `EVariantInvalidOpError` error.

Declaration: `procedure VarInvalidNullOp`

Visibility: default

Description: `VarInvalidNullOp` raises an `EVariantInvalidOpError` ([2272](#)) exception including null type description.

See also: `EVariantInvalidOpError` ([2272](#)), `VarInvalidOp` ([2262](#))

85.4.39 VarInvalidOp

Synopsis: Raise a `EVariantInvalidOpError` error.

Declaration: `procedure VarInvalidOp
procedure VarInvalidOp(const aLeft: tvartype; const aRight: tvartype;
aOpCode: tvarop)
procedure VarInvalidOp(const aRight: tvartype; aOpCode: tvarop)`

Visibility: default

Description: `VarInvalidOp` raises an `EVariantInvalidOpError` ([2272](#)) exception. when the left and/or right operand types (`aLeft`, `aRight` and the operation are specified, a description of the operand types and operation is included in the error message.

See also: `EVariantInvalidOpError` ([2272](#)), `VarCastError` ([2260](#))

85.4.40 VarIsArray

Synopsis: Check whether a variant is an array.

Declaration: `function VarIsArray(const A: Variant) : Boolean`
`function VarIsArray(const A: Variant; AResolveByRef: Boolean) : Boolean`

Visibility: default

Description: `VarIsArray` returns `True` if `A` is an array. If `AResolveByRef` is `True` (the default) then it will resolve all references first.

Errors: None.

See also: `VarTypeIsValidArrayType` (2270)

85.4.41 VarIsBool

Synopsis: Check if the variant is a boolean.

Declaration: `function VarIsBool(const V: Variant) : Boolean`

Visibility: default

Description: `VarIsCustom` returns `True` if the variant `V` is a boolean value.

See also: `VarIsEmpty` (2264), `VarIsNull` (2265), `VarIsCustom` (2264), `VarIsOrdinal` (2265), `VarIsFloat` (2265), `VarIsStr` (2266)

85.4.42 VarIsByRef

Synopsis: Check if the variant is a reference to a value.

Declaration: `function VarIsByRef(const V: Variant) : Boolean`

Visibility: default

Description: `VarIsByRef` checks whether the variant `V` is a reference to a value instead of an actual value. It returns `True` if this is the case, `False` if not.

See also: `VarType` (2270), `VarAsType` (2259), `VarIsType` (2266), `VarIsEmpty` (2264), `VarIsNull` (2265)

85.4.43 VarIsClear

Synopsis: Check if a variant is clear.

Declaration: `function VarIsClear(const V: Variant) : Boolean`

Visibility: default

Description: `VarIsClear` returns `True` if the variant `V` is empty, or it is a dispatch type with value `Nil`. For custom types, the decision is left to the custom type.

See also: `VarIsEmpty` (2264), `VarIsNull` (2265), `VarIsNumeric` (2265), `VarIsStr` (2266)

85.4.44 VarIsCustom

Synopsis: Check if the variant is a custom value.

Declaration: `function VarIsCustom(const V: Variant) : Boolean`

Visibility: default

Description: `VarIsCustom` returns `True` if the variant `V` is a custom variant (type is larger than `CFirstUserType` (2247)).

See also: `VarIsEmpty` (2264), `VarIsNull` (2265), `VarIsOrdinal` (2265), `VarIsNumeric` (2265), `VarIsStr` (2266)

85.4.45 VarIsEmpty

Synopsis: Check if the variant is empty.

Declaration: `function VarIsEmpty(const V: Variant) : Boolean`

Visibility: default

Description: `VarIsEmpty` checks whether the variant `V` is empty (i.e. the type is `varEmpty`).

See also: `VarType` (2270), `VarAsType` (2259), `VarIsType` (2266), `VarIsEmpty` (2264), `VarCheckEmpty` (2260), `VarIsNull` (2265)

85.4.46 VarIsEmptyParam

Synopsis: Check if the variant is an error value for `VAR_PARAMNOTFOUND`.

Declaration: `function VarIsEmptyParam(const V: Variant) : Boolean`

Visibility: default

Description: `VarIsEmptyParam` returns `True` if the variant `V` is an error typed variant with value `VAR_PARAMNOTFOUND`, or `False` otherwise.

See also: `VarIsError` (2264), `SetClearVarToEmptyParam` (2254)

85.4.47 VarIsError

Synopsis: Check if the variant has type `varError`.

Declaration: `function VarIsError(const V: Variant; out AResult: HRESULT) : Boolean`
`function VarIsError(const V: Variant) : Boolean`

Visibility: default

Description: `VarIsError` checks `V` and returns `True` if the type is `varError`.

See also: `VarIsNull` (2265), `VarIsEmpty` (2264), `VarIsOrdinal` (2265), `VarIsEmptyParam` (2264)

85.4.48 VarIsFloat

Synopsis: Check if the variant is a floating-point value.

Declaration: `function VarIsFloat(const V: Variant) : Boolean`

Visibility: default

Description: `VarIsCustom` returns `True` if the variant `V` is of one of the ordinal types: `FloatVarTypes` (2248) (`varSingle`, `varDouble`, `varCurrency`).

See also: `VarIsEmpty` (2264), `VarIsNull` (2265), `VarIsCustom` (2264), `VarIsOrdinal` (2265), `VarIsNumeric` (2265), `VarIsStr` (2266)

85.4.49 VarIsNull

Synopsis: Check if a variant is null.

Declaration: `function VarIsNull(const V: Variant) : Boolean`

Visibility: default

Description: `VarIsNull` returns `True` if the type of the variant `V` is `varNull`. No dereferencing is done.

See also: `VarIsEmpty` (2264), `VarType` (2270), `VarIsNumeric` (2265), `VarIsStr` (2266)

85.4.50 VarIsNumeric

Synopsis: Check if the variant is a numerical value.

Declaration: `function VarIsNumeric(const V: Variant) : Boolean`

Visibility: default

Description: `VarIsCustom` returns `True` if the variant `V` is of one of the ordinal or floating point types: `FloatVarTypes` (2248), `OrdinalVarTypes` (2248)

See also: `VarIsEmpty` (2264), `VarIsNull` (2265), `VarIsCustom` (2264), `VarIsOrdinal` (2265), `VarIsFloat` (2265), `VarIsStr` (2266)

85.4.51 VarIsOrdinal

Synopsis: Check if the variant is an ordinal value.

Declaration: `function VarIsOrdinal(const V: Variant) : Boolean`

Visibility: default

Description: `VarIsCustom` returns `True` if the variant `V` is of one of the ordinal types: `OrdinalVarTypes` (2248) (`varSmallInt`, `varInteger`, `varBoolean`, `varShortInt`, `varByte`, `varWord`, `varLongWord`, `varInt64`)

See also: `VarIsEmpty` (2264), `VarIsNull` (2265), `VarIsCustom` (2264), `VarIsFloat` (2265), `VarIsNumeric` (2265), `VarIsStr` (2266)

85.4.52 VarIsStr

Synopsis: Check if the variant is a string.

Declaration: `function VarIsStr(const V: Variant) : Boolean`

Visibility: default

Description: `VarIsCustom` returns `True` if the variant `V` is of one of the string types: `varOleStr`, `varUString`, `varString`

See also: `VarIsEmpty` (2264), `VarIsNull` (2265), `VarIsCustom` (2264), `VarIsOrdinal` (2265), `VarIsFloat` (2265), `VarIsBool` (2263)

85.4.53 VarIsType

Synopsis: Check if a variant is of certain type.

Declaration: `function VarIsType(const V: Variant; aVarType: tvartype) : Boolean`
`; Overload`
`function VarIsType(const V: Variant; const AVarTypes: Array of tvartype)`
`: Boolean; Overload`

Visibility: default

Description: `VarIsType` will return `True` if the variant `V` is of type `aVarType` or is in `aVarTypes`.

See also: `VarType` (2270), `VarAsType` (2259), `VarIsByRef` (2263), `VarIsNull` (2265), `VarIsEmpty` (2264)

85.4.54 VarNotImplError

Synopsis: Raise an `EVariantNotImplError` error.

Declaration: `procedure VarNotImplError`

Visibility: default

Description: `VarNotImplError` raises an `EVariantNotImplError` (2273) exception with a standard error message.

See also: `EVariantNotImplError` (2273), `VarInvalidNullOp` (2262), `VarInvalidOp` (2262), `VarBadIndexError` (2259), `VarArrayLockedError` (2257)

85.4.55 VarOutOfMemoryError

Synopsis: Raise an `EVariantOutOfMemoryError` error.

Declaration: `procedure VarOutOfMemoryError`

Visibility: default

Description: `VarOutOfMemoryError` raises an `EVariantOutOfMemoryError` (2273) exception with a standard error message.

See also: `EVariantOutOfMemoryError` (2273), `VarInvalidNullOp` (2262), `VarInvalidOp` (2262), `VarBadIndexError` (2259), `VarArrayLockedError` (2257)

85.4.56 VarOverflowError

Synopsis: Raise an `EVariantOverflowError` error.

Declaration: `procedure VarOverflowError`
`procedure VarOverflowError(const ASourceType: tvartype;`
`const ADestType: tvartype)`

Visibility: default

Description: `VarOverflowError` raises a `EVariantOverflowError` (2273) exception with a description of the `ASourceType` and `ADestType` types in the message.

See also: `EVariantOverflowError` (2273), `VarInvalidNullOp` (2262), `VarInvalidOp` (2262)

85.4.57 VarRangeCheckError

Synopsis: Raise an `EVariantOverflowError` error.

Declaration: `procedure VarRangeCheckError(const AType: tvartype)`
`procedure VarRangeCheckError(const ASourceType: tvartype;`
`const ADestType: tvartype)`

Visibility: default

Description: `VarRangeCheckError` raises an `EVariantOverflowError` (2273) exception with a standard error message. A description of the type `AType` is included in the error message, similarly if a `ASourceType` and `aDestType` source and destination types are provided.

See also: `EVariantOverflowError` (2273), `VarOverflowError` (2267), `VarInvalidNullOp` (2262), `VarInvalidOp` (2262), `VarBadIndexError` (2259), `VarArrayLockedError` (2257)

85.4.58 VarResultCheck

Synopsis: Check the result of an operation and raise exception if not OK.

Declaration: `procedure VarResultCheck(AResult: HRESULT)`
`procedure VarResultCheck(AResult: HRESULT; ASourceType: tvartype;`
`ADestType: tvartype)`

Visibility: default

Description: `VarResultCheck` checks the result `aResult`. If it differs from `VAR_OK` then an appropriate exception is raised based on the error code. If a source and destination type `ASourceType` and `aDestType` are specified they are included in the error message.

See also: `VarOverflowError` (2267), `VarInvalidNullOp` (2262), `VarInvalidOp` (2262), `VarBadIndexError` (2259), `VarArrayLockedError` (2257)

85.4.59 VarSameValue

Synopsis: Check if 2 variants are the same.

Declaration: `function VarSameValue(const A: Variant; const B: Variant) : Boolean`

Visibility: default

Description: `varSameValue` checks whether A and B have the same value. Here empty variants equal empty variants, and null variants equal null variants. For all other cases, the actual values are compared.

Errors: If the variants are of different types and they cannot be converted, an exception will be raised.

85.4.60 VarSupports

Synopsis: Check if a variant supports an interface.

Declaration: `function VarSupports(const V: Variant; const IID: TGuid; out Intf) : Boolean`
`function VarSupports(const V: Variant; const IID: TGuid) : Boolean`

Visibility: default

Description: `VarSupports` checks if the variant `V` contains an interface (types `Unknown`, `varDispatch`) and the interface supports the specified interface `IID`. If it does, `True` is returned, `False` otherwise. If `Intf` is specified, and the variant supports the requested interface, the interface instance is returned in `Intf`.

See also: `#rtl.sysutils.Supports` ([1779](#))

85.4.61 VarToDateTime

Synopsis: Convert a variant to datetime value.

Declaration: `function VarToDateTime(const V: Variant) : TDateTime`

Visibility: default

Description: `VarToDateTime` attempts to convert the variant `V` to a `TDateTime` value.

Errors: If the value is `Null`, an exception is raised.

See also: `VarToUnicodeStr` ([2269](#)), `VarToStrDef` ([2268](#)), `VarIsStr` ([2266](#)), `VarToWideStr` ([2269](#)), `VarToWideStrDef` ([2269](#)), `VarFromDateTime` ([2261](#))

85.4.62 VarToStr

Synopsis: Convert a variant to string value.

Declaration: `function VarToStr(const V: Variant) : AnsiString`

Visibility: default

Description: `VarToStr` attempts to convert the variant `V` to a string. If the value is `Null`, the result is an empty string.

See also: `VarToStrDef` ([2268](#)), `VarIsStr` ([2266](#)), `VarToWideStr` ([2269](#))

85.4.63 VarToStrDef

Synopsis: Convert a variant to string value, specifying a default.

Declaration: `function VarToStrDef(const V: Variant; const ADefault: AnsiString) : AnsiString`

Visibility: default

Description: `VarToStr` attempts to convert the variant `V` to a string. If the value is `Null`, the result is the specified `aDefault` string.

See also: `VarToStrDef` ([2268](#)), `VarIsStr` ([2266](#)), `VarToStr` ([2268](#))

85.4.64 VarToUnicodeStr

Synopsis: Convert a variant to string value.

Declaration: `function VarToUnicodeStr(const V: Variant) : UnicodeString`

Visibility: default

Description: `VarToUnicodeStr` attempts to convert the variant `V` to a string. If the value is `Null`, the result is an empty string.

See also: `VarToWideStr` (2269), `VarToUnicodeStrDef` (2269), `VarIsStr` (2266), `VarToStr` (2268), `VarToUnicodeStr` (2269)

85.4.65 VarToUnicodeStrDef

Synopsis: Convert a variant to string value, specifying a default.

Declaration: `function VarToUnicodeStrDef(const V: Variant;
const ADefault: UnicodeString)
: UnicodeString`

Visibility: default

Description: `VarToUnicodeStr` attempts to convert the variant `V` to a string. If the value is `Null`, the result is the specified `aDefault` string.

See also: `VarToUnicodeStr` (2269), `VarToStrDef` (2268), `VarIsStr` (2266), `VarToWideStr` (2269), `VarToWideStrDef` (2269)

85.4.66 VarToWideStr

Synopsis: Convert a variant to string value.

Declaration: `function VarToWideStr(const V: Variant) : WideString`

Visibility: default

Description: `VarToWideStr` attempts to convert the variant `V` to a string. If the value is `Null`, the result is an empty string.

See also: `VarToWideStrDef` (2269), `VarIsStr` (2266), `VarToStr` (2268), `VarToUnicodeStr` (2269)

85.4.67 VarToWideStrDef

Synopsis: Convert a variant to string value, specifying a default.

Declaration: `function VarToWideStrDef(const V: Variant; const ADefault: WideString)
: WideString`

Visibility: default

Description: `VarToWideStr` attempts to convert the variant `V` to a string. If the value is `Null`, the result is the specified `aDefault` string.

See also: `VarToWideStrDef` (2269), `VarIsStr` (2266), `VarToWideStr` (2269), `VarToUnicodeStrDef` (2269)

85.4.68 VarType

Synopsis: Return the type of a variant.

Declaration: `function VarType(const V: Variant) : tvartype`

Visibility: default

Description: `VarType` returns the type of the variant `V`. (it returns the internal type field of the variant structure)

See also: `VarTypeDeRef` ([2270](#)), `VarIsType` ([2266](#)), `VarAsType` ([2259](#))

85.4.69 VarTypeAsText

Synopsis: Return a textual description of the variant type.

Declaration: `function VarTypeAsText(const AType: tvartype) : AnsiString`

Visibility: default

Description: `VarTypeAsText` returns a textual description of the variant type `aType`. It is used in all exception raising routines to describe types, if a type is provided. For custom variant types, the name of the class is returned.

See also: `VarOpAsText` ([2248](#))

85.4.70 VarTypeDeRef

Synopsis: Returns the resolved type of a variant.

Declaration: `function VarTypeDeRef(const V: Variant) : tvartype; Overload`
`function VarTypeDeRef(const V: tvardata) : tvartype; Overload`

Visibility: default

Description: `VarTypeDeRef` will resolve the final type of the variant `V`: if `V` is a reference to another variant, the function will return the type of the referenced variant, recursing as much as needed till a non-variant type is encountered.

See also: `VarType` ([2270](#)), `VarIsType` ([2266](#)), `VarAsType` ([2259](#))

85.4.71 VarTypeIsValidArrayType

Synopsis: Check if a variant type can be used in an array.

Declaration: `function VarTypeIsValidArrayType(const aVarType: tvartype) : Boolean`

Visibility: default

Description: `VarTypeIsValidArrayType` checks if the variant type `aVarType` can be used as the type of a variant array. It returns `True` if it is usable, false otherwise. Currently, the following types are valid: `SmallInt`, `Integer`, `Single`, `Double`, `Date`, `Currency`, `OleStr`, `Dispatch`, `Error`, `Boolean`, `Variant`, `Unknown` (interface `IUnknown`), `ShortInt`, `Byte`, `Word`, `LongWord`.

See also: `VarIsArray` ([2263](#)), `VarTypeIsValidElementType` ([2271](#))

85.4.72 VarTypeIsValidElementType

Synopsis: Check if a variant type can be used in an array.

Declaration: `function VarTypeIsValidElementType(const aVarType: tvartype) : Boolean`

Visibility: default

Description: `VarTypeIsValidElementType` checks if the variant type `aVarType` can be used as the type of a variant array, similar to `VarTypeIsValidArrayType` (2270). It returns `True` if it is usable, `False` otherwise. Currently, the following types are valid: `SmallInt`, `Integer`, `Single`, `Double`, `Date`, `Currency`, `OleStr`, `Dispatch`, `Error`, `Boolean`, `Variant`, `Unknown` (interface `IUnknown`), `ShortInt`, `Byte`, `Word`, `LongWord` `int64`. In addition, if the variant type is an known custom variant type, `True` is also returned.

Reference and array indicators are stripped before the test.

See also: `VarIsArray` (2263), `VarTypeIsValidArrayType` (2270)

85.4.73 VarUnexpectedError

Synopsis: Raise an `EVariantUnexpectedError` error.

Declaration: `procedure VarUnexpectedError`

Visibility: default

Description: `VarUnexpectedError` raises an `EVariantUnexpectedError` (2274) exception with a standard error message.

See also: `EVariantUnexpectedError` (2274), `VarInvalidNullOp` (2262), `VarInvalidOp` (2262), `VarBadIndexError` (2259), `VarArrayLockedError` (2257)

85.5 EVariantArrayCreateError

85.5.1 Description

`EVariantArrayCreateError` is the exception raised when a problem is detected during creation of an array. It can be raised manually by calling `VarArrayCreateError` (2256).

See also: `VarArrayCreateError` (2256)

85.6 EVariantArrayLockedError

85.6.1 Description

`EVariantArrayLockedError` is the exception raised when the `VAR_ARRAYISLOCKED` error is encountered in `VarResultCheck` (2267)

See also: `VarResultCheck` (2267)

85.7 EVariantBadIndexError

85.7.1 Description

EVariantBadIndexError is the exception raised when the VAR_BADINDEX error is encountered in VarResultCheck ([2267](#))

See also: VarResultCheck ([2267](#))

85.8 EVariantBadVarTypeError

85.8.1 Description

EVariantBadVarTypeError is the exception raised when the VAR_BADVARTYPE error is encountered in VarResultCheck ([2267](#))

See also: VarResultCheck ([2267](#))

85.9 EVariantDispatchError

85.9.1 Description

EVariantDispatchError is the exception raised when a dispatch call fails.

See also: TCustomVariantType ([2276](#))

85.10 EVariantInvalidArgError

85.10.1 Description

EVariantInvalidArgError is the exception raised when the VAR_INVALIDARG error is encountered in VarResultCheck ([2267](#))

See also: VarResultCheck ([2267](#))

85.11 EVariantInvalidNullOpError

85.11.1 Description

EVariantInvalidNullOpError is defined for Delphi compatibility, but is not used in the FPC run time.

85.12 EVariantInvalidOpError

85.12.1 Description

EVariantInvalidOpError is the exception raised when the VAR_EXCEPTION error is encountered in VarResultCheck ([2267](#))

See also: VarResultCheck ([2267](#))

85.13 EVariantNotAnArrayError

85.13.1 Description

`EVariantNotAnArrayError` is not used in FPC and is defined for Delphi compatibility.

See also: `VarResultCheck` ([2267](#))

85.14 EVariantNotImplError

85.14.1 Description

`EVariantNotImplError` is the exception raised when the `VAR_NOTIMPL` error is encountered in `VarResultCheck` ([2267](#))

See also: `VarResultCheck` ([2267](#))

85.15 EVariantOutOfMemoryError

85.15.1 Description

`EVariantOutOfMemoryError` is the exception raised when the `VAR_OUTOFMEMORY` error is encountered in `VarResultCheck` ([2267](#)) It can be raised manually by calling `VarOutOfMemoryError` ([2266](#)).

See also: `VarOutOfMemoryError` ([2266](#)), `VarResultCheck` ([2267](#))

85.16 EVariantOverflowError

85.16.1 Description

`EVariantOverflowError` is the exception raised when the `VAR_OVERFLOW` error is encountered in `VarResultCheck` ([2267](#))

See also: `VarResultCheck` ([2267](#))

85.17 EVariantParamNotFoundError

85.17.1 Description

`EVariantParamNotFoundError` is the exception raised when the `VAR_PARAMNOTFOUND` error is encountered in `VarResultCheck` ([2267](#))

See also: `VarResultCheck` ([2267](#))

85.18 EVariantRangeCheckError

85.18.1 Description

`EVariantRangeCheckError` is defined for Delphi compatibility, it is not used in FPC.

85.19 EVariantTypeCastError

85.19.1 Description

`EVariantTypeCastError` is the exception raised when the `VAR_TYPEMISMATCH` error is encountered in `VarResultCheck` (2267)

See also: `VarResultCheck` (2267)

85.20 EVariantUnexpectedError

85.20.1 Description

`EVariantUnexpectedError` is the exception raised when the `VAR_UNEXPECTED` error is encountered in `VarResultCheck` (2267). It can be raised manually by calling `VarUnexpectedError` (2271).

See also: `VarVarUnexpectedError` (2247), `VarResultCheck` (2267)

85.21 IVarInstanceReference

85.21.1 Description

`IVarInstanceReference` is used to get the instance of an object from a custom variant. It has only one method, `GetInstance` (2274) which is used to retrieve the instance of a variant that contains an object.

The `TPublishableVariantType` (2283) descendent of `TInvokeableVariantType` (2281) uses this interface to implement reading published properties from the instance.

See also: `IVarInstanceReference.GetInstance` (2274)

85.21.2 Method overview

Page	Method	Description
2274	<code>GetInstance</code>	Return the object instance referenced by the custom variant.

85.21.3 IVarInstanceReference.GetInstance

Synopsis: Return the object instance referenced by the custom variant.

Declaration: `function GetInstance(const V: tvardata) : TObject`

Visibility: default

Description: `GetInstance` must return the object instance referenced by the custom variant `V`.

See also: `TPublishableVariantType` (2283)

85.22 IVarInvokeable

85.22.1 Description

`IVarInvokeable` must be implemented by the `TCustomVariantType` (2276) descendent if the custom variant needs to implement dynamic properties and methods.

`IVarInvokeable` has four functions that must be implemented. Any of these functions can be called whenever a custom variant's method is invoked or a property of the variant is read or written using `DispInvoke`.

The `TCustomVariantType` descendent `TInvokeableVariantType` (2281) implements the needed `DispInvoke` to call the `IVarInvokeable` interface, so you can descend from that type instead when creating a new custom variant type and override the needed functions.

See also: `TInvokeableVariantType` (2281), `TCustomVariantType` (2276)

85.22.2 Method overview

Page	Method	Description
2275	<code>DoFunction</code>	Called for methods that return a result (functions).
2275	<code>DoProcedure</code>	Called for methods that do not return a result (procedures).
2276	<code>GetProperty</code>	Called when a property must be read.
2276	<code>SetProperty</code>	Called when a property must be set.

85.22.3 IVarInvokeable.DoFunction

Synopsis: Called for methods that return a result (functions).

Declaration: `function DoFunction(var Dest: tvardata; const V: tvardata;
const Name: AnsiString;
const Arguments: TVarDataArray) : Boolean`

Visibility: default

Description: `DoFunction` is called whenever a variant method that returns a result needs to be invoked. The `Dest` parameter points to the location where the result of the function must be placed; the result must be a variant value.

`V` is the variant on which the method is being executed. The `Name` is the name of the function to execute, and `Arguments` is an array of variant arguments that were passed to the function.

The function must return `True` if the function was executed correctly, `False` otherwise.

See also: `IVarInvokeable.DoProcedure` (2275), `IVarInvokeable.GetProperty` (2276), `IVarInvokeable.SetProperty` (2276)

85.22.4 IVarInvokeable.DoProcedure

Synopsis: Called for methods that do not return a result (procedures).

Declaration: `function DoProcedure(const V: tvardata; const Name: AnsiString;
const Arguments: TVarDataArray) : Boolean`

Visibility: default

Description: `DoProcedure` is called whenever a variant method that does not return a result needs to be invoked. `V` is the variant on which the method is being executed. The `Name` is the name of the method to execute, and `Arguments` is an array of variant arguments that were passed to the method.

The function must return `True` if the procedure was executed correctly, `False` otherwise.

See also: `IVarInvokeable.DoFunction` (2275), `IVarInvokeable.GetProperty` (2276), `IVarInvokeable.SetProperty` (2276)

85.22.5 IVarInvokeable.GetProperty

Synopsis: Called when a property must be read.

Declaration: `function GetProperty(var Dest: tvardata; const V: tvardata;
const Name: AnsiString) : Boolean`

Visibility: default

Description: `DoFunction` is called whenever a variant property is read. `Dest` must be filled with the value of the property on success. `V` is the variant on which the property is read. The `Name` is the name of the property to read.

The function must return `True` if the property was read correctly, `False` otherwise.

See also: `IVarInvokeable.DoFunction` (2275), `IVarInvokeable.DoProcedure` (2275), `IVarInvokeable.SetProperty` (2276)

85.22.6 IVarInvokeable.SetProperty

Synopsis: Called when a property must be set.

Declaration: `function SetProperty(var V: tvardata; const Name: AnsiString;
const Value: tvardata) : Boolean`

Visibility: default

Description: `DoFunction` is called whenever a variant property is written. `Value` is filled with the new value of the property. `V` is the variant on which the property is written. The `Name` is the name of the property to write.

The function must return `True` if the property was written correctly, `False` otherwise.

See also: `IVarInvokeable.DoFunction` (2275), `IVarInvokeable.DoProcedure` (2275), `IVarInvokeable.GetProperty` (2276)

85.23 TCustomVariantType

85.23.1 Description

`TCustomVariantType` is used as a base class to implement custom variants. To define a custom variant type, a descendent of `TCustomVariant` must be made, and the appropriate methods must be overridden and implemented according to the specifications of the new type. Typically this means defining how your new type maps to another variant or a basic type.

Note that the `TCustomVariantType` descendent does not hold the data of the variant: it just describes how a variant record (`TVarRec` (1411)) that contains the new type's data can be examined or manipulated.

See also: `#rtl.system.TVarRec` (1411)

85.23.2 Interfaces overview

Page	Interfaces	Description
1372	IInterface	Basic interface for all COM based interfaces.

85.23.3 Method overview

Page	Method	Description
2279	BinaryOp	Perform a binary mathematical operation.
2278	Cast	Cast a custom variant to another type.
2278	CastTo	Cast a custom variant to another type.
2278	CastToOle	Cast variant value to OLE value.
2279	Clear	Clear a value.
2280	Compare	Comparison 2 custom variant values.
2280	CompareOp	Check result of a comparison between 2 custom variant values.
2279	Copy	Copy a custom variant value.
2277	Create	Instantiate a new custom variant type.
2277	Destroy	Unregister variant type.
2278	IsClear	Is the custom value unassigned ?
2280	UnaryOp	Perform a unary mathematical operation.

85.23.4 Property overview

Page	Properties	Access	Description
2281	VarType	r	Registered type.

85.23.5 TCustomVariantType.Create

Synopsis: Instantiate a new custom variant type.

Declaration: `constructor Create; Overload`
`constructor Create(RequestedVarType: tvartype); Overload`

Visibility: public

Description: `Create` sets up the necessary reference counting mechanisms to act as an interface, and assigns the new variant type identifier. If specified, the `RequestedVarType` is the variant type that is registered. if it is zero, a new identifier is assigned.

See also: `TCustomVariantType.Destroy` ([2277](#))

85.23.6 TCustomVariantType.Destroy

Synopsis: Unregister variant type.

Declaration: `destructor Destroy; Override`

Visibility: public

Description: `Destroy` unregisters the custom variant type and removes the instance from memory. After this is called, variants with the (now unregistered) type can no longer be used.

See also: `TCustomVariantType.Create` ([2277](#))

85.23.7 TCustomVariantType.IsClear

Synopsis: Is the custom value unassigned ?

Declaration: `function IsClear(const V: tvardata) : Boolean; Virtual`

Visibility: public

Description: `IsClear` must be overridden to decide whether the custom variant `V` is equivalent to `varClear` (unassigned) value. The `VarIsClear` (2263) function calls this method to decide whether a custom variant is unassigned.

The `TCustomVariantType` implementation of `IsClear` always returns `False`.

See also: `VarIsClear` (2263)

85.23.8 TCustomVariantType.Cast

Synopsis: Cast a custom variant to another type.

Declaration: `procedure Cast(var Dest: tvardata; const Source: tvardata); Virtual`

Visibility: public

Description: `Cast` is called when a variant value `Source` is cast to a custom variant `Dest`. Descendents must override this method to provide the desired conversion behaviour.

The `TCustomVariantType` implementation of `Cast` calls `CastTo` on the source variant with as destination type the custom variant type.

Errors: If the variant type cannot be cast, an exception is raised.

See also: `TCustomVariantType.CastTo` (2278), `TCustomVariantType.CastToOle` (2278)

85.23.9 TCustomVariantType.CastTo

Synopsis: Cast a custom variant to another type.

Declaration: `procedure CastTo(var Dest: tvardata; const Source: tvardata;
const aVarType: tvartype); Virtual`

Visibility: public

Description: `CastTo` is called when a custom variant value `Source` is cast to a variant type `aVarType` and stores the result in `Dest`. Descendents must override this method to provide the desired conversion behaviour.

See also: `TCustomVariantType.Cast` (2278), `TCustomVariantType.CastToOle` (2278)

85.23.10 TCustomVariantType.CastToOle

Synopsis: Cast variant value to OLE value.

Declaration: `procedure CastToOle(var Dest: tvardata; const Source: tvardata)
; Virtual`

Visibility: public

Description: `CastToOle` is called when a variant must be cast to a value that can be used in OLE variants. Descendents must override this method to provide the desired conversion behaviour.

The `TCustomVariantType` implementation of `CastToOle` raises an 'unsupported' exception.

See also: `TCustomVariantType.Cast` ([2278](#)), `TCustomVariantType.CastTo` ([2278](#))

85.23.11 `TCustomVariantType.Clear`

Synopsis: Clear a value.

Declaration: `procedure Clear(var V: tvardata); Virtual; Abstract`

Visibility: public

Description: `Clear` is called when a variant value must be cleared, by `VarClear`. This is an abstract method that must be overridden and implemented in descendent classes. When the method returns, the type of the variant record must be `varClear`

If the custom variant value used some heap memory (for instance in an object) then this is the place to release that memory.

Errors: If this method is not overridden, an abstract error will be thrown.

See also: `TCustomVariantType.IsClear` ([2278](#)), `TCustomVariantType.Copy` ([2279](#)), `VarClear` ([2260](#))

85.23.12 `TCustomVariantType.Copy`

Synopsis: Copy a custom variant value.

Declaration: `procedure Copy(var Dest: tvardata; const Source: tvardata;
const Indirect: Boolean); Virtual; Abstract`

Visibility: public

Description: `Copy` is used when a custom variant value is assigned to another variant. This is an abstract method that must be overridden and implemented in descendent classes. When the method returns, the type of the `Dest` variant record must be equal to the `Source` type (the custom variant type value).

If `Indirect` is `True` source contains a reference to the custom variant. In that case, `dest` should also contain a reference.

Errors: If this method is not overridden, an abstract error will be thrown.

See also: `TCustomVariantType.Clear` ([2279](#))

85.23.13 `TCustomVariantType.BinaryOp`

Synopsis: Perform a binary mathematical operation.

Declaration: `procedure BinaryOp(var Left: tvardata; const Right: tvardata;
const Operation: tvarop); Virtual`

Visibility: public

Description: `BinaryOp` performs `Operation` on `Left` and `Right` and stores the result of the result in `Left`. The `TCustomVariantType` implementation of this method raises an invalid operation error. Descendent classes must override this.

Errors: If a descendent does not override this, an invalid operation error is raised.

See also: `=TCustomVariantType.UnaryOp` ([2247](#)), `=TCustomVariantType.CompareOp` ([2247](#))

85.23.14 TCustomVariantType.UnaryOp

Synopsis: Perform a unary mathematical operation.

Declaration: `procedure UnaryOp(var Right: tvardata; const Operation: tvarop)
; Virtual`

Visibility: public

Description: `UnaryOp` performs `Operation` on `Right` and stores the result of the result in `Right`. The `TCustomVariantType` implementation of this method raises an invalid operation error. Descendent classes must override this method.

Errors: If a descendent does not override this, an invalid operation error is raised.

See also: `=TCustomVariantType.BinaryOp` (2247), `=TCustomVariantType.CompareOp` (2247)

85.23.15 TCustomVariantType.CompareOp

Synopsis: Check result of a comparison between 2 custom variant values.

Declaration: `function CompareOp(const Left: tvardata; const Right: tvardata;
const Operation: tvarop) : Boolean; Virtual`

Visibility: public

Description: `CompareOp` performs the comparison specified in `Operation` on `left` and `Right` and returns `True` if the values satisfy the requested operation. The `TCustomVariantType` implementation of this method raises an 'unsupported' error. Descendent classes must override this method.

Errors: If a descendent does not override this, an 'unsupported' error is raised.

See also: `TCustomVariantType.Compare` (2280), `TCustomVariantType.UnaryOp` (2280), `TCustomVariantType.binaryOp` (2279)

85.23.16 TCustomVariantType.Compare

Synopsis: Comparison 2 custom variant values.

Declaration: `procedure Compare(const Left: tvardata; const Right: tvardata;
var Relationship: TVarCompareResult); Virtual`

Visibility: public

Description: `Compare` performs the comparison specified in `Operation` on `left` and `Right` and returns the result of the comparison. The `TCustomVariantType` implementation of this method raises an 'unsupported' error. Descendent classes must override this method.

Errors: If a descendent does not override this, an 'unsupported' error is raised.

See also: `TCustomVariantType.CompareOp` (2280), `TCustomVariantType.UnaryOp` (2280), `TCustomVariantType.binaryOp` (2279)

85.23.17 TCustomVariantType.VarType

Synopsis: Registered type.

Declaration: `Property VarType : tvartype`

Visibility: public

Access: Read

Description: `VarType` is the custom variant type identifier for the custom variant values. It is assigned by the system in the constructor.

See also: `TCustomVariantType.Create` ([2277](#))

85.24 TInvokeableVariantType

85.24.1 Description

`TInvokeableVariantType` is a `TCustomVariantType` ([2276](#)) descendent which implements the `TCustomVariantType.Dispatch` ([2276](#)) method. It translates the `Dispatch` method to calls to one or more of the four methods of `IVarInvokeable` ([2275](#)): `DoFunction` ([2275](#)) or `DoProcedure` ([2275](#)) for method calls (`DISPATCH_METHOD`) and `GetProperty` ([2276](#)) for property reading (`DISPATCH_PROPERTYGET`) or `SetProperty` ([2276](#)) for property writing (`DISPATCH_PROPERTYSET`). Other combinations are possible. It has empty stubs for these methods, which must be implemented in a descendent that is used to describe a custom variant.

See also: `IVarInvokeable.DoFunction` ([2275](#)), `IVarInvokeable.DoProcedure` ([2275](#)), `IVarInvokeable.GetProperty` ([2276](#)), `IVarInvokeable.SetProperty` ([2276](#))

85.24.2 Interfaces overview

Page	Interfaces	Description
2275	<code>IVarInvokeable</code>	Interface needed for custom variants that need to implement dynamic properties and methods.

85.24.3 Method overview

Page	Method	Description
2281	<code>DoFunction</code>	Empty stub for <code>IVarInvokeable.DoFunction</code> , to be implemented in descendents.
2282	<code>DoProcedure</code>	Empty stub for <code>IVarInvokeable.DoProcedure</code> , to be implemented in descendents.
2282	<code>GetProperty</code>	Empty stub for <code>IVarInvokeable.GetProperty</code> , to be implemented in descendents.
2282	<code>SetProperty</code>	Empty stub for <code>IVarInvokeable.SetProperty</code> , to be implemented in descendents.

85.24.4 TInvokeableVariantType.DoFunction

Synopsis: Empty stub for `IVarInvokeable.DoFunction`, to be implemented in descendents.

Declaration: `function DoFunction(var Dest: tvardata; const V: tvardata; const Name: AnsiString; const Arguments: TVarDataArray) : Boolean; Virtual`

Visibility: public

Description: `DoFunction` is the default implementation of `IVarInvokeable.DoFunction` (2275), which always returns `False`. It must be overridden in descendent classes to implement actual calling of a function.

See also: `IVarInvokeable.DoFunction` (2275)

85.24.5 `TInvokeableVariantType.DoProcedure`

Synopsis: Empty stub for `IVarInvokeable.DoProcedure`, to be implemented in descendents.

Declaration: `function DoProcedure(const V: tvardata; const Name: AnsiString;
const Arguments: TVarDataArray) : Boolean; Virtual`

Visibility: public

Description: `DoProcedure` is the default implementation of `IVarInvokeable.DoProcedure` (2275), which always returns `False`. It must be overridden in descendent classes to implement actual calling of a procedure.

See also: `IVarInvokeable.DoProcedure` (2275)

85.24.6 `TInvokeableVariantType.GetProperty`

Synopsis: Empty stub for `IVarInvokeable.GetProperty`, to be implemented in descendents.

Declaration: `function GetProperty(var Dest: tvardata; const V: tvardata;
const Name: AnsiString) : Boolean; Virtual`

Visibility: public

Description: `GetProperty` is the default implementation of `IVarInvokeable.GetProperty` (2276), which always returns `False`. It must be overridden in descendent classes to implement actual reading of a property.

See also: `IVarInvokeable.GetProperty` (2276)

85.24.7 `TInvokeableVariantType.SetProperty`

Synopsis: Empty stub for `IVarInvokeable.SetProperty`, to be implemented in descendents.

Declaration: `function SetProperty(var V: tvardata; const Name: AnsiString;
const Value: tvardata) : Boolean; Virtual`

Visibility: public

Description: `SetProperty` is the default implementation of `IVarInvokeable.SetProperty` (2276), which always returns `False`. It must be overridden in descendent classes to implement actual writing of a property.

See also: `IVarInvokeable.SetProperty` (2276)

85.25 TPublishableVariantType

85.25.1 Description

TPublishableVariantType implements the IVarInvokeable.GetProperty (2276) and IVarInvokeable.SetProperty (2276) methods by getting or setting the published properties of the class instance returned by the IVarInstanceReference interface it implements.

See also: TInvokeableVariantType (2281), IVarInvokeable.GetProperty (2276), IVarInvokeable.SetProperty (2276), IVarInstanceReference (2274)

85.25.2 Interfaces overview

Page	Interfaces	Description
2274	IVarInstanceReference	Interface for variants that refer to an object.

85.25.3 Method overview

Page	Method	Description
2283	GetProperty	Read the property.
2283	SetProperty	Write the property.

85.25.4 TPublishableVariantType.GetProperty

Synopsis: Read the property.

Declaration: `function GetProperty(var Dest: tvardata; const V: tvardata;
const Name: AnsiString) : Boolean; Override`

Visibility: public

Description: GetProperty implements reading the property by looking for the property in the published properties of the instance returned by the IVarInstanceReference (2274) interface, and returning the value as a variant.

See also: IVarInstanceReference (2274), IVarInvokeable.GetProperty (2276)

85.25.5 TPublishableVariantType.SetProperty

Synopsis: Write the property.

Declaration: `function SetProperty(var V: tvardata; const Name: AnsiString;
const Value: tvardata) : Boolean; Override`

Visibility: public

Description: SetProperty implements writing the property by looking for the property in the published properties of the instance returned by the IVarInstanceReference (2274) interface, and setting the value as a variant.

See also: IVarInstanceReference (2274), IVarInvokeable.SetProperty (2276)

Chapter 86

Reference for unit 'video'

86.1 Used units

Table 86.1: Used units by unit 'video'

Name	Page
System	1335

86.2 Overview

The `Video` unit implements a screen access layer which is system independent. It can be used to write on the screen in a system-independent way, which should be optimal on all platforms for which the unit is implemented.

The working of the `Video` is simple: After calling `InitVideo` ([2301](#)), the array `VideoBuf` contains a representation of the video screen of size `ScreenWidth*ScreenHeight`, going from left to right and top to bottom when walking the array elements: `VideoBuf[0]` contains the character and color code of the top-left character on the screen. `VideoBuf[ScreenWidth]` contains the data for the character in the first column of the second row on the screen, and so on.

To write to the 'screen', the text to be written should be written to the `VideoBuf` array. Calling `UpdateScreen` ([2305](#)) will then copy the text to the screen in the most optimal way. (an example can be found further on).

The color attribute is a combination of the foreground and background color, plus the blink bit. The bits describe the various color combinations:

bits 0-3 The foreground color. Can be set using all color constants.

bits 4-6 The background color. Can be set using a subset of the color constants.

bit 7 The blinking bit. If this bit is set, the character will appear blinking.

Each possible color has a constant associated with it, see the constants section for a list of constants. The foreground and background color can be combined to a color attribute with the following code:

```
Attr:=ForeGroundColor + (BackGroundColor shl 4);
```

The color attribute can be logically or-ed with the blink attribute to produce a blinking character:

```
Attr:=Attr or blink;
```

But not all drivers may support this.

The contents of the `VideoBuf` array may be modified: This is 'writing' to the screen. As soon as everything that needs to be written in the array is in the `VideoBuf` array, calling `UpdateScreen` will copy the contents of the array screen to the screen, in a manner that is as efficient as possible.

The updating of the screen can be prohibited to optimize performance; To this end, the `LockScreenUpdate` (2302) function can be used: This will increment an internal counter. As long as the counter differs from zero, calling `UpdateScreen` (2305) will not do anything. The counter can be lowered with `UnlockScreenUpdate` (2304). When it reaches zero, the next call to `UpdateScreen` (2305) will actually update the screen. This is useful when having nested procedures that do a lot of screen writing.

The video unit also presents an interface for custom screen drivers, thus it is possible to override the default screen driver with a custom screen driver, see the `SetVideoDriver` (2304) call. The current video driver can be retrieved using the `GetVideoDriver` (2299) call.

Remark The video unit should *not* be used together with the CRT unit. Doing so will result in very strange behaviour, possibly program crashes.

86.3 Examples utility unit.

The examples in this section make use of the unit `vidutil`, which contains the `TextOut` function. This function writes a text to the screen at a given location. It looks as follows:

Listing: ./examples/videoex/vidutil.pp

```
unit vidutil;
```

```
Interface
```

```
uses
```

```
  video;
```

```
Procedure TextOut(X,Y : Word;Const S : String);
```

```
Implementation
```

```
Procedure TextOut(X,Y : Word;Const S : String);
```

```
Var
```

```
  W,P,I,M : Word;
```

```
begin
```

```
  P:=((X-1)+(Y-1)*ScreenWidth);
```

```
  M:=Length(S);
```

```
  If P+M>ScreenWidth*ScreenHeight then
```

```
    M:=ScreenWidth*ScreenHeight-P;
```

```
  For I:=1 to M do
```

```
    VideoBuf^[P+I-1]:=Ord(S[I])+($07 shl 8);
```

```
end;
```

```
end.
```

86.4 Writing a custom video driver.

Writing a custom video driver is not difficult, and generally means implementing a couple of functions, which should be registered with the `SetVideoDriver` (2304) function. The various functions that can be implemented are located in the `TVideoDriver` (2306) record:

```
TVideoDriver = Record
  InitDriver      : Procedure;
  DoneDriver      : Procedure;
  UpdateScreen    : Procedure(Force : Boolean);
  ClearScreen     : Procedure;
  SetVideoMode    : Function (Const Mode : TVideoMode) : Boolean;
  GetVideoModeCount : Function : Word;
  GetVideoModeData : Function(Index : Word; Var Data : TVideoMode) : Boolean;
  SetCursorPos    : procedure (NewCursorX, NewCursorY: Word);
  GetCursorPos    : function : Word;
  SetCursorType   : procedure (NewType: Word);
  GetCursorType   : function : Word;
end;
```

Not all of these functions must be implemented. In fact, the only absolutely necessary function to write a functioning driver is the `UpdateScreen` function. The general calls in the `Video` unit will check which functionality is implemented by the driver.

The functionality of these calls is the same as the functionality of the calls in the `video` unit, so the expected behaviour can be found in the previous section. Some of the calls, however, need some additional remarks.

InitDriver Called by `InitVideo`, this function should initialize any data structures needed for the functionality of the driver, maybe do some screen initializations. The function is guaranteed to be called only once; It can only be called again after a call to `DoneVideo`. The variables `ScreenWidth` and `ScreenHeight` should be initialized correctly after a call to this function, as the `InitVideo` call will initialize the `VideoBuf` and `OldVideoBuf` arrays based on their values.

DoneDriver This should clean up any structures that have been initialized in the `InitDriver` function. It should possibly also restore the screen as it was before the driver was initialized. The `VideoBuf` and `OldVideoBuf` arrays will be disposed of by the general `DoneVideo` call.

UpdateScreen This is the only required function of the driver. It should update the screen based on the `VideoBuf` array's contents. It can optimize this process by comparing the values with values in the `OldVideoBuf` array. After updating the screen, the `UpdateScreen` procedure should update the `OldVideoBuf` by itself. If the `Force` parameter is `True`, the whole screen should be updated, not just the changed values.

ClearScreen If there is a faster way to clear the screen than to write spaces in all character cells, then it can be implemented here. If the driver does not implement this function, then the general routines will write spaces in all video cells, and will call `UpdateScreen(True)`.

SetVideoMode Should set the desired video mode, if available. It should return `True` if the mode was set, `False` if not.

GetVideoModeCount Should return the number of supported video modes. If no modes are supported, this function should not be implemented; the general routines will return 1. (for the current mode)

GetVideoModeData Should return the data for the `Index`-th mode; `Index` is zero based. The function should return true if the data was returned correctly, false if `Index` contains an invalid index. If this is not implemented, then the general routine will return the current video mode when `Index` equals 0.

GetCapabilities If this function is not implemented, zero (i.e. no capabilities) will be returned by the general function.

The following unit shows how to override a video driver, with a driver that writes debug information to a file. The unit can be used in any of the demonstration programs, by simply including it in the `uses` clause. Setting `DetailedVideoLogging` to `True` will create a more detailed log (but will also slow down functioning)

Listing: `./examples/videoex/viddbg.pp`

```
unit viddbg;
```

```
Interface
```

```
uses video;
```

```
Procedure StartVideoLogging;
```

```
Procedure StopVideoLogging;
```

```
Function IsVideoLogging : Boolean;
```

```
Procedure SetVideoLogFileName(FileName : String);
```

```
Const
```

```
    DetailedVideoLogging : Boolean = False;
```

```
Implementation
```

```
uses sysutils,keyboard;
```

```
var
```

```
    NewVideoDriver,
    OldVideoDriver : TVideoDriver;
    Active,Logging : Boolean;
    LogFileName : String;
    VideoLog : Text;
```

```
Function TimeStamp : String;
```

```
begin
```

```
    TimeStamp:=FormatDateTime('hh:nn:ss',Time());
```

```
end;
```

```
Procedure StartVideoLogging;
```

```
begin
```

```
    Logging:=True;
```

```
    Writeln(VideoLog,'Start logging video operations at: ',TimeStamp);
```

```
end;
```

```
Procedure StopVideoLogging;
```

```
begin
```

```
    Writeln(VideoLog,'Stop logging video operations at: ',TimeStamp);
```

```
    Logging:=False;
```



```

end;

Function IsVideoLogging : Boolean;

begin
  IsVideoLogging:=Logging;
end;

Var
  ColUpd,RowUpd : Array[0..1024] of Integer;

Procedure DumpScreenStatistics(Force : Boolean);

Var
  I,Count : Integer;

begin
  If Force then
    Write(VideoLog,'forced ');
    Writeln(VideoLog,'video update at ',TimeStamp,' : ');
    FillChar(Colupd,SizeOf(ColUpd),#0);
    FillChar(Rowupd,SizeOf(RowUpd),#0);
    Count:=0;
    For I:=0 to VideoBufSize div SizeOf(TVideoCell) do
      begin
        If VideoBuf^[I]<>OldVideoBuf^[I] then
          begin
            Inc(Count);
            Inc(ColUpd[I mod ScreenWidth]);
            Inc(RowUpd[I div ScreenHeight]);
          end;
        end;
      Write(VideoLog,Count,' videocells differed divided over ');
      Count:=0;
      For I:=0 to ScreenWidth-1 do
        If ColUpd[I]<>0 then
          Inc(Count);
      Write(VideoLog,Count,' columns and ');
      Count:=0;
      For I:=0 to ScreenHeight-1 do
        If RowUpd[I]<>0 then
          Inc(Count);
      Writeln(VideoLog,Count,' rows. ');
      If DetailedVideoLogging Then
        begin
          For I:=0 to ScreenWidth-1 do
            If (ColUpd[I]<>0) then
              Writeln(VideoLog,'Col ',I,' : ',ColUpd[I]:3,' rows changed');
          For I:=0 to ScreenHeight-1 do
            If (RowUpd[I]<>0) then
              Writeln(VideoLog,'Row ',I,' : ',RowUpd[I]:3,' columns changed');
          end;
        end;
      end;

Procedure LogUpdateScreen(Force : Boolean);

begin
  If Logging then

```

```

    DumpScreenStatistics(Force);
    OldVideoDriver.UpdateScreen(Force);
end;

Procedure LogInitVideo;

begin
    OldVideoDriver.InitDriver();
    Assign(VideoLog,logFileName);
    Rewrite(VideoLog);
    Active:=True;
    StartVideoLogging;
end;

Procedure LogDoneVideo;

begin
    StopVideoLogging;
    Close(VideoLog);
    Active:=False;
    OldVideoDriver.DoneDriver();
end;

Procedure SetVideoLogFileName(FileName : String);

begin
    If Not Active then
        LogFileName:=FileName;
    end;

Initialization
    GetVideoDriver(OldVideoDriver);
    NewVideoDriver:=OldVideoDriver;
    NewVideoDriver.UpdateScreen:=@LogUpdateScreen;
    NewVideoDriver.InitDriver:=@LogInitVideo;
    NewVideoDriver.DoneDriver:=@LogDoneVideo;
    LogFileName:='Video.log';
    Logging:=False;
    SetVideoDriver(NewVideoDriver);
end.

```

86.5 Constants, types and variables

86.5.1 Constants

Black = 0

Black color attribute.

Blink = 128

Blink attribute.

Blue = 1

Blue color attribute.

Brown = 6

Brown color attribute.

cpBlink = \$0002

Video driver supports blink attribute.

cpChangeCursor = \$0020

Video driver supports changing cursor shape.

cpChangeFont = \$0008

Video driver supports changing screen font.

cpChangeMode = \$0010

Video driver supports changing mode.

cpColor = \$0004

Video driver supports color.

cpUnderLine = \$0001

Video driver supports underline attribute.

crBlock = 2

Block cursor.

crHalfBlock = 3

Half block cursor.

crHidden = 0

Hide cursor.

crUnderLine = 1

Underline cursor.

Cyan = 3

Cyan color attribute.

DarkGray = 8

Dark gray color attribute.

```
errOk = 0
```

No error.

```
ErrorCode : LongInt = ErrOK
```

Error code returned by the last operation.

```
ErrorHandler : TErrorHandler = @ DefaultErrorHandler
```

The `ErrorHandler` variable can be set to a custom-error handling function. It is set by default to the `DefaultErrorHandler` (2296) function.

```
ErrorInfo : Pointer = Nil
```

Pointer to extended error information.

```
errVioBase = 1000
```

Base value for video errors.

```
errVioInit = errVioBase + 1
```

Video driver initialization error.

```
errVioNoSuchMode = errVioBase + 3
```

Invalid video mode.

```
errVioNotSupported = errVioBase + 2
```

Unsupported video function.

```
FVMaxWidth = 240
```

Maximum screen buffer width.

```
Green = 2
```

Green color attribute.

```
iso_codepages = [iso01, iso02, iso03, iso04, iso05, iso06, iso07,
  iso08, iso09, iso10, iso13, iso14, iso15]
```

`iso_codepages` is a set containing all code pages that use an ISO encoding.

```
LightBlue = 9
```

Light Blue color attribute.

LightCyan = 11

Light cyan color attribute.

LightGray = 7

Light gray color attribute.

LightGreen = 10

Light green color attribute.

LightMagenta = 13

Light magenta color attribute.

LightRed = 12

Light red color attribute.

LowAscii = True

On some systems, the low 32 values of the DOS code page are necessary for the ASCII control codes and cannot be displayed by programs. If LowAscii is true, you can use the low 32 ASCII values. If it is false, you must avoid using them.

LowAscii can be implemented either through a constant, variable or property. You should under no circumstances assume that you can write to LowAscii, or take its address.

Magenta = 5

Magenta color attribute.

NoExtendedFrame = False

The VT100 character set only has line drawing characters consisting of a single line. If this value is true, the line drawing characters with two lines will be automatically converted to single lines.

NoExtendedFrame can be implemented either through a constant, variable or property. You should under no circumstances assume that you can write to NoExtendedFrame, or take its address.

Red = 4

Red color attribute.

ScreenHeight : Word = 0

Current screen height.

ScreenWidth : Word = 0

Current screen Width.

```
vga_codepages = [cp437, cp850, cp852, cp866]
```

`vga_codepages` is a set containing all code pages that can be considered a normal vga font (as in use on early VGA cards) Note that KOI8-R has line drawing characters in wrong place.

```
vioOK = 0
```

No errors occurred.

```
White = 15
```

White color attribute.

```
Yellow = 14
```

Yellow color attribute.

86.5.2 Types

```
PVideoBuf = ^TVideoBuf
```

Pointer type to `TVideoBuf` (2294).

```
PVideoCell = ^TVideoCell
```

Pointer type to `TVideoCell` (2294).

```
PVideoMode = ^TVideoMode
```

Pointer to `TVideoMode` (2306) record.

```
TErrorHandler = function(Code: LongInt; Info: Pointer)
    : TErrorHandlerReturnValue
```

The `TErrorHandler` function is used to register an own error handling function. It should be used when installing a custom error handling function, and must return one of the above values.

`Code` should contain the error code for the error condition, and the `Info` parameter may contain any data type specific to the error code passed to the function.

```
TErrorHandlerReturnValue = (errRetry, errAbort, errContinue)
```

Table 86.2: Enumeration values for type `TErrorHandlerReturnValue`

Value	Explanation
<code>errAbort</code>	abort and return error code.
<code>errContinue</code>	abort without returning an errorcode.
<code>errRetry</code>	retry the operation.

Type used to report and respond to error conditions.

```
TVideoBuf = Array[0..32759] of TVideoCell
```

The TVideoBuf type represents the screen.

```
TVideoCell = Word
```

TVideoCell describes one character on the screen. One of the bytes contains the color attribute with which the character is drawn on the screen, and the other byte contains the ASCII code of the character to be drawn. The exact position of the different bytes in the record is operating system specific. On most little-endian systems, the high byte represents the color attribute, while the low-byte represents the ASCII code of the character to be drawn.

```
TVideoModeSelector = function(const VideoMode: TVideoMode;
    Params: LongInt) : Boolean
```

Video mode selection callback prototype.

86.5.3 Variables

```
CursorLines : Byte
```

CursorLines is a bitmask which determines which cursor lines are visible and which are not. Each set bit corresponds to a cursorline being shown.

This variable is not supported on all platforms, so it should be used sparingly.

```
CursorX : Word
```

Current horizontal position in the screen where items will be written.

```
CursorY : Word
```

Current vertical position in the screen where items will be written.

```
external_codepage : Tencoding
```

This variable is for internal use only and should not be used.

```
internal_codepage : Tencoding
```

This variable is for internal use only and should not be used.

```
OldVideoBuf : PVideoBuf
```

The OldVideoBuf contains the state of the video screen after the last screen update. The Update-Screen (2305) function uses this array to decide which characters on screen should be updated, and which not.

Note that the OldVideoBuf array may be ignored by some drivers, so it should not be used. The Array is in the interface section of the video unit mainly so drivers that need it can make use of it.

```
ScreenColor : Boolean
```

ScreenColor indicates whether the current screen supports colors.

VideoBuf : PVideoBuf

VideoBuf forms the heart of the Video unit: This variable represents the physical screen. Writing to this array and calling UpdateScreen (2305) will write the actual characters to the screen.

VideoBufSize : LongInt

Current size of the video buffer pointed to by VideoBuf (2295)

86.6 Procedures and functions

86.6.1 ClearScreen

Synopsis: Clear the video screen.

Declaration: `procedure ClearScreen`

Visibility: default

Description: `ClearScreen` clears the entire screen, and calls `UpdateScreen` (2305) after that. This is done by writing spaces to all character cells of the video buffer in the default color (lightgray on black, color attribute \07).

Errors: None.

See also: `InitVideo` (2301), `UpdateScreen` (2305)

Listing: ./examples/videoex/ex3.pp

```

program testvideo;

uses video,keyboard,vidutil;

Var
  i : longint;
  k : TkeyEvent;

begin
  InitVideo;
  InitKeyboard;
  For i:=1 to 10 do
    TextOut(i,i, 'Press any key to clear screen');
    UpdateScreen(false);
    K:=GetKeyEvent;
    ClearScreen;
    TextOut(1,1,'Cleared screen. Press any key to end');
    UpdateScreen(true);
    K:=GetKeyEvent;
    DoneKeyBoard;
    DoneVideo;
end.

```

86.6.2 DefaultErrorHandler

Synopsis: Default error handling routine.

Declaration: `function DefaultErrorHandler (AErrorCode: LongInt; AErrorInfo: Pointer)
: TErrorHandlerReturnValue`

Visibility: default

Description: `DefaultErrorHandler` is the default error handler used by the video driver. It simply sets the error code `AErrorCode` and `AErrorInfo` in the global variables `ErrorCode` and `ErrorInfo` and returns `errContinue`.

Errors: None.

86.6.3 DoneVideo

Synopsis: Disable video driver.

Declaration: `procedure DoneVideo`

Visibility: default

Description: `DoneVideo` disables the Video driver if the video driver is active. If the videodriver was already disabled or not yet initialized, it does nothing. Disabling the driver means it will clean up any allocated resources, possibly restore the screen in the state it was before `InitVideo` was called. Particularly, the `VideoBuf` and `OldVideoBuf` arrays are no longer valid after a call to `DoneVideo`.

The `DoneVideo` should always be called if `InitVideo` was called. Failing to do so may leave the screen in an unusable state after the program exits.

For an example, see most other functions.

Errors: Normally none. If the driver reports an error, this is done through the `ErrorCode` variable.

See also: `InitVideo` ([2301](#))

86.6.4 GetCapabilities

Synopsis: Get current driver capabilities.

Declaration: `function GetCapabilities : Word`

Visibility: default

Description: `GetCapabilities` returns the capabilities of the current driver. It is an or-ed combination of the following constants:

cpUnderLine Video driver supports underline attribute.

cpBlink Video driver supports blink attribute.

cpColor Video driver supports color.

cpChangeFont Video driver supports changing screen font.

cpChangeMode Video driver supports changing mode.

cpChangeCursor Video driver supports changing cursor shape.

Note that the video driver should not yet be initialized to use this function. It is a property of the driver.

Errors: None.

See also: [GetCursorType \(2297\)](#), [GetVideoDriver \(2299\)](#)

Listing: ./examples/videoex/ex4.pp

Program Example4;

{ Program to demonstrate the GetCapabilities function. }

Uses video;

Var

W: Word;

Procedure TestCap(Cap: Word; Msg : **String**);

begin

Write(Msg, ' : ');

If (W **and** Cap=Cap) **then**

Writeln('Yes')

else

Writeln('No');

end;

begin

 W:=GetCapabilities;

Writeln('Video driver supports following functionality');

 TestCap(cpUnderLine,'Underlined characters');

 TestCap(cpBlink,'Blinking characters');

 TestCap(cpColor,'Color characters');

 TestCap(cpChangeFont,'Changing font');

 TestCap(cpChangeMode,'Changing video mode');

 TestCap(cpChangeCursor,'Changing cursor shape');

end.

86.6.5 GetCursorType

Synopsis: Get screen cursor type.

Declaration: `function GetCursorType : Word`

Visibility: default

Description: `GetCursorType` returns the current cursor type. It is one of the following values:

crHiddenHide cursor.

crUnderLineUnderline cursor.

crBlockBlock cursor.

crHalfBlockHalf block cursor.

Note that not all drivers support all types of cursors.

Errors: None.

See also: [SetCursorType \(2303\)](#), [GetCapabilities \(2296\)](#)

Listing: ./examples/videoex/ex5.pp

Program Example5;

{ Program to demonstrate the GetCursorType function. }

Uses video,keyboard,vidutil;

Const

CursorTypes : **Array**[crHidden..crHalfBlock] **of string** =
('Hidden','UnderLine','Block','HalfBlock');

begin

InitVideo;
InitKeyboard;
TextOut(1,1,'Cursor type: '+CursorTypes[GetCursorType]);
TextOut(1,2,'Press any key to exit.');

UpdateScreen(False);

GetKeyEvent;

DoneKeyboard;

DoneVideo;

end.

86.6.6 GetLockScreenCount

Synopsis: Get the screen lock update count.

Declaration: `function GetLockScreenCount : Integer`

Visibility: default

Description: `GetLockScreenCount` returns the current lock level. When the lock level is zero, a call to `UpdateScreen` (2305) will actually update the screen.

Errors: None.

See also: `LockScreenUpdate` (2302), `UnlockScreenUpdate` (2304), `UpdateScreen` (2305)

Listing: ./examples/videoex/ex6.pp

Program Example6;

{ Program to demonstrate the GetLockScreenCount function. }

Uses video,keyboard,vidutil;

Var

I : Longint;

S : **String**;

begin

InitVideo;

InitKeyboard;

TextOut(1,1,'Press key till new text appears.');

UpdateScreen(False);

Randomize;

For I:=0 **to** Random(10)+1 **do**

LockScreenUpdate;

```

l:=0;
While GetLockScreenCount<>0 do
  begin
    Inc(l);
    Str(l,S);
    UnlockScreenUpdate;
    GetKeyEvent;
    TextOut(1,1,'UnlockScreenUpdate had to be called '+S+' times');
    UpdateScreen(False);
  end;
  TextOut(1,2,'Press any key to end. ');
  UpdateScreen(False);
  GetKeyEvent;
  DoneKeyboard;
  DoneVideo;
end.

```

86.6.7 GetVideoDriver

Synopsis: Get a copy of the current video driver.

Declaration: `procedure GetVideoDriver(var Driver: TVideoDriver)`

Visibility: default

Description: `GetVideoDriver` returns the currently active video driver record in `Driver`. It can be used to clone the current video driver, or to override certain parts of it using the `SetVideoDriver` (2304) call.

Errors: None.

See also: `SetVideoDriver` (2304)

86.6.8 GetVideoMode

Synopsis: Return current video mode.

Declaration: `procedure GetVideoMode(var Mode: TVideoMode)`

Visibility: default

Description: `GetVideoMode` returns the settings of the currently active video mode. The `row`, `col` fields indicate the dimensions of the current video mode, and `Color` is true if the current video supports colors.

See also: `SetVideoMode` (2304), `GetVideoModeData` (2301)

Listing: `./examples/videoex/ex7.pp`

Program Example7;

{ Program to demonstrate the GetVideoMode function. }

Uses video,keyboard,vidutil;

Var

 M : TVideoMode;

 S : **String**;

```

begin
  InitVideo;
  InitKeyboard;
  GetVideoMode(M);
  if M.Color then
    TextOut(1,1,'Current mode has color')
  else
    TextOut(1,1,'Current mode does not have color');
  Str(M.Row,S);
  TextOut(1,2,'Number of rows : '+S);
  Str(M.Col,S);
  TextOut(1,3,'Number of columns : '+S);
  Textout(1,4,'Press any key to exit. ');
  UpdateScreen(False);
  GetKeyEvent;
  DoneKeyboard;
  DoneVideo;
end.

```

86.6.9 GetVideoModeCount

Synopsis: Get the number of video modes supported by the driver.

Declaration: `function GetVideoModeCount : Word`

Visibility: default

Description: `GetVideoModeCount` returns the number of video modes that the current driver supports. If the driver does not support switching of modes, then 1 is returned.

This function can be used in conjunction with the `GetVideoModeData` (2301) function to retrieve data for the supported video modes.

Errors: None.

See also: `GetVideoModeData` (2301), `GetVideoMode` (2299)

Listing: `./examples/videoex/ex8.pp`

Program Example8;

{ Program to demonstrate the GetVideoModeCount function. }

Uses video,keyboard,vidutil;

Procedure DumpMode (M : TVideoMode; Index : Integer);

Var

S : String;

begin

Str(Index:2,S);

inc(Index);

TextOut(1,Index,'Data for mode '+S+' ');

if M.Color then

TextOut(19,Index,' color,')

else

```

    TextOut(19,Index,'No color,');
    Str(M.Row:3,S);
    TextOut(28,Index,S+' rows');
    Str(M.Col:3,S);
    TextOut(36,Index,S+' columns');
end;

Var
  i,Count : Integer;
  m : TVideoMode;

begin
  InitVideo;
  InitKeyboard;
  Count:=GetVideoModeCount;
  For I:=1 to Count do
    begin
      GetVideoModeData(I-1,M);
      DumpMode(M,I-1);
    end;
    TextOut(1,Count+1,'Press any key to exit');
    UpdateScreen(False);
    GetKeyEvent;
    DoneKeyboard;
    DoneVideo;
  end.

```

86.6.10 GetVideoModeData

Synopsis: Get the specifications for a video mode.

Declaration: `function GetVideoModeData(Index: Word; var Data: TVideoMode) : Boolean`

Visibility: default

Description: `GetVideoModeData` returns the characteristics of the `Index`-th video mode in `Data`. `Index` is zero based, and has a maximum value of `GetVideoModeCount-1`. If the current driver does not support setting of modes (`GetVideoModeCount=1`) and `Index` is zero, the current mode is returned.

The function returns `True` if the mode data was retrieved successfully, `False` otherwise.

For an example, see `GetVideoModeCount` ([2300](#)).

Errors: In case `Index` has a wrong value, `False` is returned.

See also: `GetVideoModeCount` ([2300](#)), `SetVideoMode` ([2304](#)), `GetVideoMode` ([2299](#))

86.6.11 InitVideo

Synopsis: Initialize video driver.

Declaration: `procedure InitVideo`

Visibility: default

Description: `InitVideo` initializes the video subsystem. If the video system was already initialized, it does nothing. After the driver has been initialized, the `VideoBuf` and `OldVideoBuf` pointers are

initialized, based on the `ScreenWidth` and `ScreenHeight` variables. When this is done, the screen is cleared.

For an example, see most other functions.

Errors: if the driver fails to initialize, the `ErrorCode` variable is set.

See also: `DoneVideo` ([2296](#))

86.6.12 LockScreenUpdate

Synopsis: Prevent further screen updates.

Declaration: `procedure LockScreenUpdate`

Visibility: `default`

Description: `LockScreenUpdate` increments the screen update lock count with one. As long as the screen update lock count is not zero, `UpdateScreen` ([2305](#)) will not actually update the screen.

This function can be used to optimize screen updating: If a lot of writing on the screen needs to be done (by possibly unknown functions), calling `LockScreenUpdate` before the drawing, and `UnlockScreenUpdate` ([2304](#)) after the drawing, followed by a `UpdateScreen` ([2305](#)) call, all writing will be shown on screen at once.

For an example, see `GetLockScreenCount` ([2298](#)).

Errors: None.

See also: `UpdateScreen` ([2305](#)), `UnlockScreenUpdate` ([2304](#)), `GetLockScreenCount` ([2298](#))

86.6.13 SetCursorPos

Synopsis: Set write cursor position.

Declaration: `procedure SetCursorPos (NewCursorX: Word; NewCursorY: Word)`

Visibility: `default`

Description: `SetCursorPos` positions the cursor on the given position: Column `NewCursorX` and row `NewCursorY`. The origin of the screen is the upper left corner, and has coordinates `(0, 0)`.

The current position is stored in the `CursorX` and `CursorY` variables.

Errors: None.

See also: `SetCursorType` ([2303](#))

Listing: `./examples/videoex/ex2.pp`

program example2;

uses video,keyboard;

Var

 P,PP,D : Integer;
 K: TKeyEvent;

Procedure PutSquare (P : Integer; C : Char);

begin

```

VideoBuf^[P]:=Ord(C)+($07 shl 8);
VideoBuf^[P+ScreenWidth]:=Ord(c)+($07 shl 8);
VideoBuf^[P+1]:=Ord(c)+($07 shl 8);
VideoBuf^[P+ScreenWidth+1]:=Ord(c)+($07 shl 8);
end;

begin
InitVideo;
InitKeyBoard;
P:=0;
PP:=-1;
Repeat
  If PP<>-1 then
    PutSquare(PP, ' ');
    PutSquare(P, '#');
    SetCursorPos(P Mod ScreenWidth, P div ScreenWidth);
    UpdateScreen(False);
    PP:=P;
  Repeat
    D:=0;
    K:=TranslateKeyEvent(GetKeyEvent);
    Case GetKeyEventCode(K) of
      kbdLeft : If (P Mod ScreenWidth)<>0 then
        D:=-1;
      kbdUp : If P>=ScreenWidth then
        D:=-ScreenWidth;
      kbdRight : If ((P+2) Mod ScreenWidth)<>0 then
        D:=1;
      kbdDown : if (P<(VideoBufSize div 2)-(ScreenWidth*2)) then
        D:=ScreenWidth;
    end;
  Until (D<>0) or (GetKeyEventChar(K)='q');
  P:=P+D;
until GetKeyEventChar(K)='q';
DoneKeyBoard;
DoneVideo;
end.

```

86.6.14 SetCursorType

Synopsis: Set cursor type.

Declaration: `procedure SetCursorType(NewType: Word)`

Visibility: default

Description: `SetCursorType` sets the cursor to the type specified in `NewType`.

crHidden Hide cursor.

crUnderLine Underline cursor.

crBlock Block cursor.

crHalfBlock Half block cursor.

Errors: None.

See also: `SetCursorPos` ([2302](#))

86.6.15 SetVideoDriver

Synopsis: Install a new video driver.

Declaration: `function SetVideoDriver(const Driver: TVideoDriver) : Boolean`

Visibility: default

Description: `SetVideoDriver` sets the videodriver to be used to `Driver`. If the current videodriver is initialized (after a call to `InitVideo`) then it does nothing and returns `False`.

A new driver can only be installed if the previous driver was not yet activated (i.e. before a call to `InitVideo` (2301)) or after it was deactivated (i.e after a call to `DoneVideo`).

For more information about installing a videodriver, see `viddriver` (2286).

For an example, see the section on writing a custom video driver.

Errors: If the current driver is initialized, then `False` is returned.

See also: `viddriver` (2286)

86.6.16 SetVideoMode

Synopsis: Set current video mode.

Declaration: `function SetVideoMode(const Mode: TVideoMode) : Boolean`

Visibility: default

Description: `SetVideoMode` sets the video mode to the mode specified in `Mode`:

If the call was successful, then the screen will have `Col` columns and `Row` rows, and will be displaying in color if `Color` is `True`.

The function returns `True` if the mode was set successfully, `False` otherwise.

Note that the video mode may not always be set. E.g. a console on Linux or a telnet session cannot always set the mode. It is important to check the error value returned by this function if it was not successful.

The mode can be set when the video driver has not yet been initialized (i.e. before `InitVideo` (2301) was called) In that case, the video mode will be stored, and after the driver was initialized, an attempt will be made to set the requested mode. Changing the video driver before the call to `InitVideo` will clear the stored video mode.

To know which modes are valid, use the `GetVideoModeCount` (2300) and `GetVideoModeData` (2301) functions. To retrieve the current video mode, use the `GetVideoMode` (2299) procedure.

Errors: If the specified mode cannot be set, then `errVioNoSuchMode` may be set in `ErrorCode`

See also: `GetVideoModeCount` (2300), `GetVideoModeData` (2301), `GetVideoMode` (2299)

86.6.17 UnlockScreenUpdate

Synopsis: Unlock screen update.

Declaration: `procedure UnlockScreenUpdate`

Visibility: default

Description: `UnlockScreenUpdate` decrements the screen update lock count with one if it is larger than zero. When the lock count reaches zero, the `UpdateScreen` (2305) will actually update the screen. No screen update will be performed as long as the screen update lock count is nonzero. This mechanism can be used to increase screen performance in case a lot of writing is done.

It is important to make sure that each call to `LockScreenUpdate` (2302) is matched by exactly one call to `UnlockScreenUpdate`

For an example, see `GetLockScreenCount` (2298).

Errors: None.

See also: `LockScreenUpdate` (2302), `GetLockScreenCount` (2298), `UpdateScreen` (2305)

86.6.18 UpdateScreen

Synopsis: Update physical screen with internal screen image.

Declaration: `procedure UpdateScreen(Force: Boolean)`

Visibility: default

Description: `UpdateScreen` synchronizes the actual screen with the contents of the `VideoBuf` internal buffer. The parameter `Force` specifies whether the whole screen has to be redrawn (`Force=True`) or only parts that have changed since the last update of the screen.

The `Video` unit keeps an internal copy of the screen as it last wrote it to the screen (in the `OldVideoBuf` array). The current contents of `VideoBuf` are examined to see what locations on the screen need to be updated. On slow terminals (e.g. a Linux telnet session) this mechanism can speed up the screen redraw considerably.

On platforms where mouse cursor visibility is not guaranteed to be preserved during screen updates this routine has to restore the mouse cursor after the update (usually by calling `HideMouse` from unit `Mouse` before the real update and `ShowMouse` afterwards).

For an example, see most other functions.

Errors: None.

See also: `ClearScreen` (2295)

86.7 TVideoDriver

```
TVideoDriver = record
  InitDriver : procedure;
  DoneDriver : procedure
;
  UpdateScreen : procedure(Force: Boolean);
  ClearScreen : procedure
;
  SetVideoMode : function(const Mode: TVideoMode) : Boolean;
  GetVideoModeCount
  : function : Word;
  GetVideoModeData : function(Index: Word; var
  Data: TVideoMode) : Boolean;
  SetCursorPos : procedure(NewCursorX
  : Word; NewCursorY: Word);
  GetCursorType : function : Word;
```

```
SetCursorType
: procedure (NewType: Word);
GetCapabilities : function : Word;
end
```

`TVideoDriver` record can be used to install a custom video driver, with the `SetVideoDriver` ([2304](#)) call.

An explanation of all fields can be found there.

86.8 TVideoMode

```
TVideoMode = record
  Col : Word;
  Row : Word;
  Color : Boolean
;
end
```

The `TVideoMode` record describes a videomode. Its fields are self-explaining: `Col`, `Row` describe the number of columns and rows on the screen for this mode. `Color` is `True` if this mode supports colors, or `False` if not.

Chapter 87

Reference for unit 'WinCRT'

87.1 Used units

Table 87.1: Used units by unit 'WinCRT'

Name	Page
System	1335

87.2 Overview

The `wincrt` unit provides some auxiliary routines for use with the `graph` ([884](#)) unit, namely keyboard support. It has no connection with the `crt` ([583](#)) unit, nor with the Turbo-Pascal for Windows `WinCrt` unit. As such, it should not be used by end users. Refer to the `crt` ([583](#)) unit instead.

87.3 Constants, types and variables

87.3.1 Variables

`directvideo` : Boolean

On windows, this variable is ignored.

`lastmode` : Word

Is supposed to contain the last used video mode, but is actually unused.

87.4 Procedures and functions

87.4.1 `delay`

Synopsis: Pause program execution.

Declaration: `procedure delay(ms: Word)`

Visibility: default

Description: Delay stops program execution for the indicated number ms of milliseconds.

See also: sound ([2308](#)), nosound ([2308](#))

87.4.2 keypressed

Synopsis: Check if a key was pressed.

Declaration: `function keypressed : Boolean`

Visibility: default

Description: KeyPressed returns True if the user pressed a key, or False if not. It does not wait for the user to press a key.

See also: readkey ([2308](#))

87.4.3 nosound

Synopsis: Stop the speaker.

Declaration: `procedure nosound`

Visibility: default

Description: NoSound does nothing, windows does not support this.

See also: sound ([2308](#))

87.4.4 readkey

Synopsis: Read a key from the keyboard.

Declaration: `function readkey : Char`

Visibility: default

Description: ReadKey reads a key from the keyboard, and returns the ASCII value of the key, or the scancode of the key in case it is a special key.

The function waits until a key is pressed.

See also: KeyPressed ([2308](#))

87.4.5 sound

Synopsis: Sound PC speaker.

Declaration: `procedure sound(hz: Word)`

Visibility: default

Description: Sound sounds the PC speaker. It emits a tone with frequency Hz for 500 milliseconds. (the time argument is required by the windows API)

See also: nosound ([2308](#))

87.4.6 textmode

Synopsis: Set indicated text mode.

Declaration: `procedure textmode(mode: Integer)`

Visibility: `default`

Description: `TextMode` does nothing.

Chapter 88

Reference for unit 'WinDirs'

88.1 Used units

Table 88.1: Used units by unit 'WinDirs'

Name	Page
System	1335

88.2 Overview

This unit contains only one function: `GetWindowsSpecialDir` ([2324](#)). It is a simple pascal wrapper around the `shfolder.dll` library, and is available under Windows only.

88.3 Constants, types and variables

88.3.1 Constants

`CSIDL_ADMINTOOLS` = \$0030

ID for the personal "Start Menu\Programs\Administrative tools" folder (In the user's personal folder).

`CSIDL_APPDATA` = \$001A

ID for the roaming "Application Data" folder (in the user's personal folder).

`CSIDL_CDBURN_AREA` = \$003B

ID for the personal "Local Settings\Application Data\Microsoft\CD Burning" folder (Under "All users" folder).

`CSIDL_COMMON_ADMINTOOLS` = \$002F

ID for the common "Start Menu\Programs\Administrative tools" folder (Under "All users" folder).

CSIDL_COMMON_APPDATA = \$0023

ID for the common "Application Data" folder (Under "All users" folder).

CSIDL_COMMON_DESKTOPDIRECTORY = \$0019

ID for the public "Desktop" folder (Under "All users" folder).

CSIDL_COMMON_DOCUMENTS = \$002E

ID for the common "Documents" folder (Under "All users" folder).

CSIDL_COMMON_FAVORITES = \$001F

ID for the public "Favorites" folder (Under "All users" folder).

CSIDL_COMMON_MUSIC = \$0035

ID for the common "Documents\My Music" folder (Under "All users" folder).

CSIDL_COMMON_PICTURES = \$0036

ID for the common "Documents\My Pictures" folder (Under "All users" folder).

CSIDL_COMMON_PROGRAMS = \$0017

ID for the public "Programs" folder (Under "All users" Start menu folder).

CSIDL_COMMON_STARTMENU = \$0016

ID for the public "Start Menu" folder (Under "All users").

CSIDL_COMMON_STARTUP = \$0018

ID for the public "Startup" folder (Under "All users\Start menu\Programs" folder).

CSIDL_COMMON_TEMPLATES = \$002D

ID for the common "Templates" folder (Under "All users" folder).

CSIDL_COMMON_VIDEO = \$0037

ID for the common "Documents\My Video" folder (Under "All users" folder).

CSIDL_COOKIES = \$0021

ID for the "Cookies" folder (in the user's personal folder).

CSIDL_DESKTOPDIRECTORY = \$0010

ID for the "Desktop" folder (in the user's personal folder).

CSIDL_FAVORITES = \$0006

ID for the "Favourites" folder (in the user's personal folder).

CSIDL_FLAG_CREATE = \$8000

Flag to specify if the requested folder must be created if it didn't exist yet.

CSIDL_FONTS = \$0014

CSIDL_HISTORY = \$0022

ID for the "History" folder (in the user's personal folder).

CSIDL_INTERNET_CACHE = \$0020

ID for the "Temporary Internet Files" folder (in the user's personal folder).

CSIDL_LOCAL_APPDATA = \$001C

ID for the local "Application Data" folder (in the user's personal folder).

CSIDL_MYMUSIC = \$000D

ID for the "My Music" folder (in the user's personal folder).

CSIDL_MYPICTURES = \$0027

ID for the "My Pictures" folder (in the user's personal folder).

CSIDL_MYVIDEO = \$000E

ID for the "My Videos" folder (in the user's personal folder).

CSIDL_NETHOOD = \$0013

ID for the "Nethood" folder (in the user's personal folder).

CSIDL_PERSONAL = \$0005

ID for the "My Documents" folder (in the user's personal folder).

CSIDL_PRINTHOOD = \$001B

ID for the "Printhood" folder (in the user's personal folder).

CSIDL_PROFILE = \$0028

ID for the user's personal folder.

CSIDL_PROFILES = \$003E

ID for the parent folder for the user's folders.

CSIDL_PROGRAMS = \$0002

ID for the "Program files" folder.

CSIDL_PROGRAM_FILES = \$0026

ID for the "Program Files" folder (alias for CSIDL_PROGRAMS).

CSIDL_PROGRAM_FILES_COMMON = \$002B

ID for the Common folder under the "Program Files" folder.

CSIDL_RECENT = \$0008

ID for the "Recent" folder (in the user's personal folder).

CSIDL_SENDTO = \$0009

ID for the "Send to" folder (in the user's personal folder).

CSIDL_STARTMENU = \$000B

ID for the "Start menu" folder (in the user's personal folder).

CSIDL_STARTUP = \$0007

ID for the "Startup" menu folder (in the user's personal folder).

CSIDL_SYSTEM = \$0025

ID for the Windows System32 dir.

CSIDL_TEMPLATES = \$0015

ID for the "Templates" folder (in the user's personal folder).

CSIDL_WINDOWS = \$0024

ID for the Windows installation dir.

FOLDERID_AccountPictures : TGuid =
'{008CA0B1-55B4-4C56-B8A8-4DE4B299D3BE}'

FOLDERID_AddNewPrograms : TGuid =
'{DE61D971-5EBC-4F02-A3A9-6C82895E5C04}'

FOLDERID_AdminTools : TGuid =
'{724EF170-A42D-4FEF-9F26-B60E846FBA4F}'

FOLDERID_AllAppMods : TGuid =
'{7AD67899-66AF-43BA-9156-6AAD42E6C596}'

FOLDERID_AppCaptures : TGuid =
'{EDC0FE71-98D8-4F4A-B920-C8DC133CB165}'

FOLDERID_AppDataDesktop : TGuid =
'{B2C5E279-7ADD-439F-B28C-C41FE1BBF672}'

FOLDERID_AppDataDocuments : TGuid =
'{7BE16610-1F7F-44AC-BFF0-83E15F2FFCA1}'

FOLDERID_AppDataFavorites : TGuid =
'{7CFBEFBC-DE1F-45AA-B843-A542AC536CC9}'

FOLDERID_AppDataProgramData : TGuid =
'{559D40A3-A036-40FA-AF61-84CB430A4D34}'

FOLDERID_ApplicationShortcuts : TGuid =
'{A3918781-E5F2-4890-B3D9-A7E54332328C}'

FOLDERID_AppsFolder : TGuid =
'{1E87508D-89C2-42F0-8A7E-645A0F50CA58}'

FOLDERID_AppUpdates : TGuid =
'{A305CE99-F527-492B-8B1A-7E76FA98D6E4}'

FOLDERID_CameraRoll : TGuid =
'{AB5FB87B-7CE2-4F83-915D-550846C9537B}'

FOLDERID_CameraRollLibrary : TGuid =
'{2B20DF75-1EDA-4039-8097-38798227D5B7}'

FOLDERID_CDBurning : TGuid =
'{9E52AB10-F80D-49DF-ACB8-4330F5687855}'

FOLDERID_ChangeRemovePrograms : TGuid =
'{DF7266AC-9274-4867-8D55-3BD661DE872D}'

FOLDERID_CommonAdminTools : TGuid =
'{D0384E7D-BAC3-4797-8F14-CBA229B392B5}'

FOLDERID_CommonOEMLinks : TGuid =
'{C1BAE2D0-10DF-4334-BEDD-7AA20B227A9D}'

FOLDERID_CommonPrograms : TGuid =
'{0139D44E-6AFE-49F2-8690-3DAFCAE6FFB8}'

FOLDERID_CommonStartMenu : TGuid =
'{A4115719-D62E-491D-AA7C-E74B8BE3B067}'

FOLDERID_CommonStartMenuPlaces : TGuid =
'{A440879F-87A0-4F7D-B700-0207B966194A}'

FOLDERID_CommonStartup : TGuid =
'{82A5EA35-D9CD-47C5-9629-E15D2F714E6E}'

FOLDERID_CommonTemplates : TGuid =
'{B94237E7-57AC-4347-9151-B08C6C32D1F7}'

FOLDERID_ComputerFolder : TGuid =
'{0AC0837C-BBF8-452A-850D-79D08E667CA7}'

FOLDERID_ConflictFolder : TGuid =
'{4BFEFB45-347D-4006-A5BE-AC0CB0567192}'

FOLDERID_ConnectionsFolder : TGuid =
'{6F0CD92B-2E97-45D1-88FF-B0D186B8DEDD}'

FOLDERID_Contacts : TGuid =
'{56784854-C6CB-462B-8169-88E350ACB882}'

FOLDERID_ControlPanelFolder : TGuid =
'{82A74AEB-AEB4-465C-A014-D097EE346D63}'

FOLDERID_Cookies : TGuid =
'{2B0F765D-C0E9-4171-908E-08A611B84FF6}'

FOLDERID_CurrentAppMods : TGuid =
'{3DB40B20-2A30-4DBE-917E-771DD21DD099}'

FOLDERID_Desktop : TGuid =
'{B4BFCC3A-DB2C-424C-B029-7FE99A87C641}'

FOLDERID_DevelopmentFiles : TGuid =
'{DBE8E08E-3053-4BBC-B183-2A7B2B191E59}'

FOLDERID_Device : TGuid =
'{1C2AC1DC-4358-4B6C-9733-AF21156576F0}'

FOLDERID_DeviceMetadataStore : TGuid =
'{5CE4A5E9-E4EB-479D-B89F-130C02886155}'

FOLDERID_Documents : TGuid =
'{FDD39AD0-238F-46AF-ADB4-6C85480369C7}'

FOLDERID_DocumentsLibrary : TGuid =
'{7B0DB17D-9CD2-4A93-9733-46CC89022E7C}'

FOLDERID_Downloads : TGuid =
'{374DE290-123F-4565-9164-39C4925E467B}'

FOLDERID_Favorites : TGuid =
'{1777F761-68AD-4D8A-87BD-30B759FA33DD}'

FOLDERID_Fonts : TGuid = '{FD228CB7-AE11-4AE3-864C-16F3910AB8FE}'

FOLDERID_Games : TGuid = '{CAC52C1A-B53D-4EDC-92D7-6B2E8AC19434}'

FOLDERID_GameTasks : TGuid =
'{054FAE61-4DD8-4787-80B6-090220C4B700}'

FOLDERID_History : TGuid =
'{D9DC8A3B-B784-432E-A781-5A1130A75963}'

FOLDERID_HomeGroup : TGuid =
'{52528A6B-B9E3-4ADD-B60D-588C2DBA842D}'

FOLDERID_HomeGroupCurrentUser : TGuid =
'{9B74B6A3-0DFD-4F11-9E78-5F7800F2E772}'

```
FOLDERID_ImplicitAppShortcuts : TGuid =
    '{BCB5256F-79F6-4CEE-B725-DC34E402FD46}'

FOLDERID_InternetCache : TGuid =
    '{352481E8-33BE-4251-BA85-6007CAEDCF9D}'

FOLDERID_InternetFolder : TGuid =
    '{4D9F7874-4E0C-4904-967B-40B0D20C3E4B}'

FOLDERID_Libraries : TGuid =
    '{1B3EA5DC-B587-4786-B4EF-BD1DC332AEAE}'

FOLDERID_Links : TGuid = '{BFB9D5E0-C6A9-404C-B2B2-AE6DB6AF4968}'

FOLDERID_LocalAppData : TGuid =
    '{F1B32785-6FBA-4FCF-9D55-7B8E7F157091}'

FOLDERID_LocalAppDataLow : TGuid =
    '{A520A1A4-1780-4FF6-BD18-167343C5AF16}'

FOLDERID_LocalDocuments : TGuid =
    '{F42EE2D3-909F-4907-8871-4C22FC0BF756}'

FOLDERID_LocalDownloads : TGuid =
    '{7D83EE9B-2244-4E70-B1F5-5393042AF1E4}'

FOLDERID_LocalizedResourcesDir : TGuid =
    '{2A00375E-224C-49DE-B8D1-440DF7EF3DDC}'

FOLDERID_LocalMusic : TGuid =
    '{A0C69A99-21C8-4671-8703-7934162FCF1D}'

FOLDERID_LocalPictures : TGuid =
    '{0DDD015D-B06C-45D5-8C4C-F59713854639}'

FOLDERID_LocalVideos : TGuid =
    '{35286A68-3C57-41A1-BBB1-0EAE73D76C95}'

FOLDERID_Music : TGuid = '{4BD8D571-6D19-48D3-BE97-422220080E43}'
```

```

FOLDERID_MusicLibrary : TGuid =
    '{2112AB0A-C86A-4FFE-A368-0DE96E47012E}'

FOLDERID_NetHood : TGuid =
    '{C5ABBF53-E17F-4121-8900-86626FC2C973}'

FOLDERID_NetworkFolder : TGuid =
    '{D20BEEC4-5CA8-4905-AE3B-BF251EA09B53}'

FOLDERID_Objects3D : TGuid =
    '{31C0DD25-9439-4F12-BF41-7FF4EDA38722}'

FOLDERID_OneDrive : TGuid =
    '{A52BBA46-E9E1-435F-B3D9-28DAA648C0F6}'

FOLDERID_OriginalImages : TGuid =
    '{2C36C0AA-5812-4B87-BFD0-4CD0DFB19B39}'

FOLDERID_PhotoAlbums : TGuid =
    '{69D2CF90-FC33-4FB7-9A0C-EBB0F0FCB43C}'

FOLDERID_Pictures : TGuid =
    '{33E28130-4E1E-4676-835A-98395C3BC3BB}'

FOLDERID_PicturesLibrary : TGuid =
    '{A990AE9F-A03B-4E80-94BC-9912D7504104}'

FOLDERID_Playlists : TGuid =
    '{DE92C1C7-837F-4F69-A3BB-86E631204A23}'

FOLDERID_PrintersFolder : TGuid =
    '{76FC4E2D-D6AD-4519-A663-37BD56068185}'

FOLDERID_PrintHood : TGuid =
    '{9274BD8D-CFD1-41C3-B35E-B13F55A758F4}'

FOLDERID_Profile : TGuid =
    '{5E6C858F-0E22-4760-9AFE-EA3317B67173}'

FOLDERID_ProgramData : TGuid =
    '{62AB5D82-FDC1-4DC3-A9DD-070D1D495D97}'

```

```
FOLDERID_ProgramFiles : TGuid =
    '{905E63B6-C1BF-494E-B29C-65B732D3D21A}'
```

```
FOLDERID_ProgramFilesCommon : TGuid =
    '{F7F1ED05-9F6D-47A2-AAAE-29D317C6F066}'
```

```
FOLDERID_ProgramFilesCommonX64 : TGuid =
    '{6365D5A7-0F0D-45E5-87F6-0DA56B6A4F7D}'
```

```
FOLDERID_ProgramFilesCommonX86 : TGuid =
    '{DE974D24-D9C6-4D3E-BF91-F4455120B917}'
```

```
FOLDERID_ProgramFilesX64 : TGuid =
    '{6D809377-6AF0-444B-8957-A3773F02200E}'
```

```
FOLDERID_ProgramFilesX86 : TGuid =
    '{7C5A40EF-A0FB-4BFC-874A-C0F2E0B9FA8E}'
```

```
FOLDERID_Programs : TGuid =
    '{A77F5D77-2E2B-44C3-A6A2-ABA601054A51}'
```

```
FOLDERID_Public : TGuid =
    '{DFDF76A2-C82A-4D63-906A-5644AC457385}'
```

```
FOLDERID_PublicDesktop : TGuid =
    '{C4AA340D-F20F-4863-AFEF-F87EF2E6BA25}'
```

```
FOLDERID_PublicDocuments : TGuid =
    '{ED4824AF-DCE4-45A8-81E2-FC7965083634}'
```

```
FOLDERID_PublicDownloads : TGuid =
    '{3D644C9B-1FB8-4F30-9B45-F670235F79C0}'
```

```
FOLDERID_PublicGameTasks : TGuid =
    '{DEBF2536-E1A8-4C59-B6A2-414586476AEA}'
```

```
FOLDERID_PublicLibraries : TGuid =
    '{48DAF80B-E6CF-4F4E-B800-0E69D84EE384}'
```

```
FOLDERID_PublicMusic : TGuid =
    '{3214FAB5-9757-4298-BB61-92A9DEAA44FF}'
```



```
FOLDERID_PublicPictures : TGuid =
    '{B6EBFB86-6907-413C-9AF7-4FC2ABF07CC5}'
```

```
FOLDERID_PublicRingtones : TGuid =
    '{E555AB60-153B-4D17-9F04-A5FE99FC15EC}'
```

```
FOLDERID_PublicUserTiles : TGuid =
    '{0482AF6C-08F1-4C34-8C90-E17EC98B1E17}'
```

```
FOLDERID_PublicVideos : TGuid =
    '{2400183A-6185-49FB-A2D8-4A392A602BA3}'
```

```
FOLDERID_QuickLaunch : TGuid =
    '{52A4F021-7B75-48A9-9F6B-4B87A210BC8F}'
```

```
FOLDERID_Recent : TGuid =
    '{AE50C081-EBD2-438A-8655-8A092E34987A}'
```

```
FOLDERID_RecordedCalls : TGuid =
    '{2F8B40C2-83ED-48EE-B383-A1F157EC6F9A}'
```

```
FOLDERID_RecordedTVLibrary : TGuid =
    '{1A6FDBA2-F42D-4358-A798-B74D745926C5}'
```

```
FOLDERID_RecycleBinFolder : TGuid =
    '{B7534046-3ECB-4C18-BE4E-64CD4CB7D6AC}'
```

```
FOLDERID_ResourceDir : TGuid =
    '{8AD10C31-2ADB-4296-A8F7-E4701232C972}'
```

```
FOLDERID_RetailDemo : TGuid =
    '{12D4C69E-24AD-4923-BE19-31321C43A767}'
```

```
FOLDERID_Ringtones : TGuid =
    '{C870044B-F49E-4126-A9C3-B52A1FF411E8}'
```

```
FOLDERID_RoamedTileImages : TGuid =
    '{AAA8D5A5-F1D6-4259-BAA8-78E7EF60835E}'
```

```
FOLDERID_RoamingAppData : TGuid =
    '{3EB685DB-65F9-4CF6-A03A-E3EF65729F3D}'
```

```

FOLDERID_RoamingTiles : TGuid =
    '{00BCFC5A-ED94-4E48-96A1-3F6217F21990}'

FOLDERID_SampleMusic : TGuid =
    '{B250C668-F57D-4EE1-A63C-290EE7D1AA1F}'

FOLDERID_SamplePictures : TGuid =
    '{C4900540-2379-4C75-844B-64E6FAF8716B}'

FOLDERID_SamplePlaylists : TGuid =
    '{15CA69B3-30EE-49C1-ACE1-6B5EC372AFB5}'

FOLDERID_SampleVideos : TGuid =
    '{859EAD94-2E85-48AD-A71A-0969CB56A6CD}'

FOLDERID_SavedGames : TGuid =
    '{4C5C32FF-BB9D-43B0-B5B4-2D72E54EAAA4}'

FOLDERID_SavedPictures : TGuid =
    '{3B193882-D3AD-4EAB-965A-69829D1FB59F}'

FOLDERID_SavedPicturesLibrary : TGuid =
    '{E25B5812-BE88-4BD9-94B0-29233477B6C3}'

FOLDERID_SavedSearches : TGuid =
    '{7D1D3A04-DEBB-4115-95CF-2F29DA2920DA}'

FOLDERID_Screenshots : TGuid =
    '{B7BEDE81-DF94-4682-A7D8-57A52620B86F}'

FOLDERID_SearchHistory : TGuid =
    '{0D4C3DB6-03A3-462F-A0E6-08924C41B5D4}'

FOLDERID_SearchHome : TGuid =
    '{190337D1-B8CA-4121-A639-6D472D16972A}'

FOLDERID_SearchTemplates : TGuid =
    '{7E636BFE-DFA9-4D5E-B456-D7B39851D8A9}'

FOLDERID_SEARCH_CSC : TGuid =
    '{EE32E446-31CA-4ABA-814F-A5EBD2FD6D5E}'

```

FOLDERID_SEARCH_MAPI : TGuid =
'{98EC0E18-2098-4D44-8644-66979315A281}'

FOLDERID_SendTo : TGuid =
'{8983036C-27C0-404B-8F08-102D10DCFD74}'

FOLDERID_SidebarDefaultParts : TGuid =
'{7B396E54-9EC5-4300-BE0A-2482EBAE1A26}'

FOLDERID_SidebarParts : TGuid =
'{A75D362E-50FC-4FB7-AC2C-A8BEAA314493}'

FOLDERID_SkyDrive : TGuid =
'{A52BBA46-E9E1-435F-B3D9-28DAA648C0F6}'

FOLDERID_SkyDriveCameraRoll : TGuid =
'{767E6811-49CB-4273-87C2-20F355E1085B}'

FOLDERID_SkyDriveDocuments : TGuid =
'{24D89E24-2F19-4534-9DDE-6A6671FBB8FE}'

FOLDERID_SkyDriveMusic : TGuid =
'{C3F2459E-80D6-45DC-BFEF-1F769F2BE730}'

FOLDERID_SkyDrivePictures : TGuid =
'{339719B5-8C47-4894-94C2-D8F77ADD44A6}'

FOLDERID_StartMenu : TGuid =
'{625B53C3-AB48-4EC1-BA1F-A1EF4146FC19}'

FOLDERID_StartMenuAllPrograms : TGuid =
'{F26305EF-6948-40B9-B255-81453D09C785}'

FOLDERID_Startup : TGuid =
'{B97D20BB-F46A-4C97-BA10-5E3608430854}'

FOLDERID_SyncManagerFolder : TGuid =
'{43668BF8-C14E-49B2-97C9-747784D784B7}'

FOLDERID_SyncResultsFolder : TGuid =
'{289A9A43-BE44-4057-A41B-587A76D7E7F9}'

```
FOLDERID_SyncSetupFolder : TGuid =
    '{0F214138-B1D3-4A90-BBA9-27CBC0C5389A}'
```

```
FOLDERID_System : TGuid =
    '{1AC14E77-02E7-4E5D-B744-2EB1AE5198B7}'
```

```
FOLDERID_SystemX86 : TGuid =
    '{D65231B0-B2F1-4857-A4CE-A8E7C6EA7D27}'
```

```
FOLDERID_Templates : TGuid =
    '{A63293E8-664E-48DB-A079-DF759E0509F7}'
```

```
FOLDERID_UserPinned : TGuid =
    '{9E3995AB-1F9C-4F13-B827-48B24B6C7174}'
```

```
FOLDERID_UserProfiles : TGuid =
    '{0762D272-C50A-4BB0-A382-697DCD729B80}'
```

```
FOLDERID_UserProgramFiles : TGuid =
    '{5CD7AEE2-2219-4A67-B85D-6C9CE15660CB}'
```

```
FOLDERID_UserProgramFilesCommon : TGuid =
    '{BCBD3057-CA5C-4622-B42D-BC56DB0AE516}'
```

```
FOLDERID_UsersFiles : TGuid =
    '{F3CE0F7C-4901-4ACC-8648-D5D44B04EF8F}'
```

```
FOLDERID_UsersLibraries : TGuid =
    '{A302545D-DEFF-464B-ABE8-61C8648D939B}'
```

```
FOLDERID_Videos : TGuid =
    '{18989B1D-99B5-455B-841C-AB7C74E4DDFC}'
```

```
FOLDERID_VideosLibrary : TGuid =
    '{491E922F-5643-4AF4-A7EB-4E7A138D8174}'
```

```
FOLDERID_Windows : TGuid =
    '{F38BF404-1D43-42F2-9305-67DE0B28FC23}'
```

```
KF_FLAG_CREATE = DWORD($8000)
```

```
KF_FLAG_DEFAULT = DWORD(0)
```

88.4 Procedures and functions

88.4.1 ConvertCSIDLtoFOLDERID

Declaration: `function ConvertCSIDLtoFOLDERID(CSIDL: Integer; out FOLDERID: TGuid)
: Boolean`

Visibility: default

88.4.2 ConvertFOLDERIDtoCSIDL

Declaration: `function ConvertFOLDERIDtoCSIDL(const FOLDERID: TGuid;
out CSIDL: Integer) : Boolean`

Visibility: default

88.4.3 GetWindowsSpecialDir

Synopsis: Get the location of a special directory.

Declaration: `function GetWindowsSpecialDir(ID: Integer; CreateIfNotExists: Boolean)
: string
function GetWindowsSpecialDir(const GUID: TGuid;
CreateIfNotExists: Boolean) : string`

Visibility: default

Description: `GetWindowsSpecialDir` can be used to get the path of special folders on the Windows system. The locations are identified using one of the `CSIDL_*` constants. If the ID of a location is or-ed with the `CSIDL_FLAG_CREATE` constant, then the directory will be created if it didn't exist yet (and the user has sufficient rights to do so).

Errors: On error, an empty string is returned.

88.4.4 GetWindowsSpecialDirUnicode

Declaration: `function GetWindowsSpecialDirUnicode(ID: Integer;
CreateIfNotExists: Boolean)
: UnicodeString
function GetWindowsSpecialDirUnicode(const GUID: TGuid;
CreateIfNotExists: Boolean)
: UnicodeString`

Visibility: default

88.4.5 GetWindowsSystemDirectory

Declaration: `function GetWindowsSystemDirectory : string`

Visibility: default

88.4.6 GetWindowsSystemDirectoryUnicode

Declaration: `function GetWindowsSystemDirectoryUnicode : UnicodeString`

Visibility: `default`

Chapter 89

Reference for unit 'x86'

89.1 Used units

Table 89.1: Used units by unit 'x86'

Name	Page
BaseUnix	155
System	1335

89.2 Overview

The x86 unit contains some of the routines that were present in the 1.0.X Linux unit, and which were Intel (PC) architecture specific.

These calls have been preserved for compatibility, but should be considered deprecated: they are not portable and may not even work on future Linux versions.

89.3 Procedures and functions

89.3.1 fpIOperm

Synopsis: Set permission on IO ports.

Declaration: `function fpIOperm(From: Cardinal; Num: Cardinal; Value: cint) : cint`

Visibility: default

Description: `fpIOperm` sets permissions on `Num` ports starting with port `From` to `Value`. The function returns zero if the call was successful, a nonzero value otherwise.

Note:

- This works ONLY as root.
- Only the first `0x03ff` ports can be set.
- When doing a `FpFork` ([204](#)), the permissions are reset. When doing a `FpExecVE` ([201](#)) they are kept.

Errors: Extended error information can be retrieved with `FpGetErrno` ([207](#))

89.3.2 fpIoPL

Synopsis: Set I/O privilege level.

Declaration: `function fpIoPL(Level: cint) : cint`

Visibility: default

Description: `FpIoPL` sets the I/O privilege level. It is intended for completeness only, one should normally not use it.

89.3.3 ReadPort

Synopsis: Read data from a PC port.

Declaration: `procedure ReadPort(Port: LongInt; var Value: Byte)`
`procedure ReadPort(Port: LongInt; var Value: LongInt)`
`procedure ReadPort(Port: LongInt; var Value: Word)`

Visibility: default

Description: `ReadPort` reads one Byte, Word or Longint from port `Port` into `Value`.

Note that you need permission to read a port. This permission can be set by the root user with the `FpIOPerm` (2326) call.

Errors: In case of an error (not enough permissions read this port), runtime 216 (*Access Violation*) will occur.

See also: `FpIOPerm` (2326), `ReadPortB` (2327), `ReadPortW` (2328), `ReadPortL` (2328), `WritePort` (2328), `WritePortB` (2329), `WritePortL` (2329), `WritePortW` (2329)

89.3.4 ReadPortB

Synopsis: Read bytes from a PC port.

Declaration: `function ReadPortB(Port: LongInt) : Byte`
`procedure ReadPortB(Port: LongInt; var Buf; Count: LongInt)`

Visibility: default

Description: The procedural form of `ReadPortB` reads `Count` bytes from port `Port` and stores them in `Buf`. There must be enough memory allocated at `Buf` to store `Count` bytes.

The functional form of `ReadPortB` reads 1 byte from port `B` and returns the byte that was read.

Note that you need permission to read a port. This permission can be set by the root user with the `FpIOPerm` (2326) call.

Errors: In case of an error (not enough permissions read this port), runtime 216 (*Access Violation*) will occur.

See also: `FpIOPerm` (2326), `ReadPort` (2327), `ReadPortW` (2328), `ReadPortL` (2328), `WritePort` (2328), `WritePortB` (2329), `WritePortL` (2329), `WritePortW` (2329)

89.3.5 ReadPortL

Synopsis: Read longints from a PC port.

Declaration: `function ReadPortL(Port: LongInt) : LongInt`
`procedure ReadPortL(Port: LongInt; var Buf; Count: LongInt)`

Visibility: default

Description: The procedural form of `ReadPortL` reads `Count` longints from port `Port` and stores them in `Buf`. There must be enough memory allocated at `Buf` to store `Count` Longints.

The functional form of `ReadPortL` reads 1 longint from port `B` and returns the longint that was read.

Note that you need permission to read a port. This permission can be set by the root user with the `FpIOPerm` (2326) call.

Errors: In case of an error (not enough permissions read this port), runtime 216 (*Access Violation*) will occur.

See also: `FpIOPerm` (2326), `ReadPort` (2327), `ReadPortW` (2328), `ReadPortB` (2327), `WritePort` (2328), `WritePortB` (2329), `WritePortL` (2329), `WritePortW` (2329)

89.3.6 ReadPortW

Synopsis: Read Words from a PC port.

Declaration: `function ReadPortW(Port: LongInt) : Word`
`procedure ReadPortW(Port: LongInt; var Buf; Count: LongInt)`

Visibility: default

Description: The procedural form of `ReadPortW` reads `Count` words from port `Port` and stores them in `Buf`. There must be enough memory allocated at `Buf` to store `Count` words.

The functional form of `ReadPortW` reads 1 word from port `B` and returns the word that was read.

Note that you need permission to read a port. This permission can be set by the root user with the `FpIOPerm` (2326) call.

Errors: In case of an error (not enough permissions read this port), runtime 216 (*Access Violation*) will occur.

See also: `FpIOPerm` (2326), `ReadPort` (2327), `ReadPortB` (2327), `ReadPortL` (2328), `WritePort` (2328), `WritePortB` (2329), `WritePortL` (2329), `WritePortW` (2329)

89.3.7 WritePort

Synopsis: Write data to PC port.

Declaration: `procedure WritePort(Port: LongInt; Value: Byte)`
`procedure WritePort(Port: LongInt; Value: LongInt)`
`procedure WritePort(Port: LongInt; Value: Word)`

Visibility: default

Description: `WritePort` writes `Value` – 1 byte, `Word` or longint – to port `Port`.

Remark You need permission to write to a port. This permission can be set with root permission with the `FpIOPerm` (2326) call.

Errors: In case of an error (not enough permissions to write to this port), runtime 216 (*Access Violation*) will occur.

See also: [FpIOPerm \(2326\)](#), [WritePortB \(2329\)](#), [WritePortL \(2329\)](#), [WritePortW \(2329\)](#), [ReadPortB \(2327\)](#), [ReadPortL \(2328\)](#), [ReadPortW \(2328\)](#)

89.3.8 WritePortB

Synopsis: Write byte to PC port.

Declaration: `procedure WritePortB(Port: LongInt; Value: Byte)`
`procedure WritePortB(Port: LongInt; var Buf; Count: LongInt)`

Visibility: default

Description: The first form of `WritePortB` writes 1 byte to port `Port`. The second form writes `Count` bytes from `Buf` to port `Port`.

Remark You need permission to write to a port. This permission can be set with root permission with the [FpIOPerm \(2326\)](#) call.

Errors: In case of an error (not enough permissions to write to this port), runtime 216 (*Access Violation*) will occur.

See also: [FpIOPerm \(2326\)](#), [WritePort \(2328\)](#), [WritePortL \(2329\)](#), [WritePortW \(2329\)](#), [ReadPortB \(2327\)](#), [ReadPortL \(2328\)](#), [ReadPortW \(2328\)](#)

89.3.9 WritePortL

Synopsis: Write longint to PC port.

Declaration: `procedure WritePortL(Port: LongInt; Value: LongInt)`
`procedure WritePortL(Port: LongInt; var Buf; Count: LongInt)`

Visibility: default

Description: The first form of `WritePortL` writes 1 longint to port `Port`. The second form writes `Count` longints from `Buf` to port `Port`.

Remark You need permission to write to a port. This permission can be set with root permission with the [FpIOPerm \(2326\)](#) call.

Errors: In case of an error (not enough permissions to write to this port), runtime 216 (*Access Violation*) will occur.

See also: [FpIOPerm \(2326\)](#), [WritePort \(2328\)](#), [WritePortB \(2329\)](#), [WritePortW \(2329\)](#), [ReadPortB \(2327\)](#), [ReadPortL \(2328\)](#), [ReadPortW \(2328\)](#)

89.3.10 WritePortW

Synopsis: Write Word to PC port.

Declaration: `procedure WritePortW(Port: LongInt; Value: Word)`
`procedure WritePortW(Port: LongInt; var Buf; Count: LongInt)`

Visibility: default

Description: The first form of `WritePortB` writes 1 byte to port `Port`. The second form writes `Count` bytes from `Buf` to port `Port`.

Remark You need permission to write to a port. This permission can be set with root permission with the `FpIOPerm` (2326) call.

Errors: In case of an error (not enough permissions to write to this port), runtime 216 (*Access Violation*) will occur.

See also: `FpIOPerm` (2326), `WritePort` (2328), `WritePortL` (2329), `WritePortB` (2329), `ReadPortB` (2327), `ReadPortL` (2328), `ReadPortW` (2328)