

Run-Time Library (RTL) :
Reference guide.

Free Pascal version 3.0.0:
Reference guide for RTL units.
Document version 2.6
June 2015

Michaël Van Canneyt

Contents

0.1	Overview	99
1	Reference for unit 'BaseUnix'	100
1.1	Used units	100
1.2	Overview	100
1.3	Constants, types and variables	100
1.3.1	Constants	100
1.3.2	Types	123
1.4	Procedures and functions	140
1.4.1	CreateShellArgV	140
1.4.2	FpAccess	141
1.4.3	FpAlarm	141
1.4.4	FpChdir	142
1.4.5	FpChmod	142
1.4.6	FpChown	144
1.4.7	FpClose	145
1.4.8	FpClosedir	145
1.4.9	FpDup	145
1.4.10	FpDup2	146
1.4.11	FpExecv	147
1.4.12	FpExecve	148
1.4.13	FpExit	149
1.4.14	FpFcntl	149
1.4.15	fpdffillset	150
1.4.16	fpFD_CLR	150
1.4.17	fpFD_ISSET	151
1.4.18	fpFD_SET	151
1.4.19	fpFD_ZERO	151
1.4.20	FpFork	151
1.4.21	FPFStat	152
1.4.22	FpFtruncate	153

1.4.23	FpGetcwd	153
1.4.24	FpGetegid	153
1.4.25	FpGetEnv	154
1.4.26	fpgeterno	154
1.4.27	FpGeteuid	155
1.4.28	FpGetgid	155
1.4.29	FpGetgroups	156
1.4.30	FpGetpggrp	156
1.4.31	FpGetpid	156
1.4.32	FpGetppid	157
1.4.33	fpGetPriority	157
1.4.34	FpGetRLimit	157
1.4.35	FpGetsid	158
1.4.36	FpGetuid	158
1.4.37	FpIOCtl	159
1.4.38	FpKill	159
1.4.39	FpLink	160
1.4.40	FpLseek	161
1.4.41	fpLstat	161
1.4.42	FpMkdir	163
1.4.43	FpMkfifo	163
1.4.44	Fpmmmap	163
1.4.45	Fpmunmap	165
1.4.46	FpNanoSleep	165
1.4.47	fpNice	166
1.4.48	FpOpen	167
1.4.49	FpOpendir	168
1.4.50	FpPause	169
1.4.51	FpPipe	169
1.4.52	FpPoll	170
1.4.53	FppRead	170
1.4.54	FppWrite	171
1.4.55	FpRead	171
1.4.56	FpReaddir	172
1.4.57	fpReadLink	173
1.4.58	FpReadV	174
1.4.59	FpRename	174
1.4.60	FpRmdir	175
1.4.61	fpSelect	175
1.4.62	fpseterno	177

1.4.63	FpSetgid	177
1.4.64	fpSetPriority	177
1.4.65	FpSetRLimit	178
1.4.66	FpSetsid	178
1.4.67	fpsettimeofday	178
1.4.68	FpSetuid	178
1.4.69	FPSigaction	179
1.4.70	FpSigAddSet	180
1.4.71	FpSigDelSet	180
1.4.72	FpsigEmptySet	180
1.4.73	FpSigFillSet	181
1.4.74	FpSigIsMember	181
1.4.75	FpSignal	181
1.4.76	FpSigPending	182
1.4.77	FpSigProcMask	182
1.4.78	FpSigSuspend	183
1.4.79	FpSigTimedWait	183
1.4.80	FpSleep	183
1.4.81	FpStat	184
1.4.82	fpSymlink	185
1.4.83	fpS_ISBLK	186
1.4.84	fpS_ISCHR	186
1.4.85	fpS_ISDIR	187
1.4.86	fpS_ISFIFO	187
1.4.87	fpS_ISLNK	187
1.4.88	fpS_ISREG	188
1.4.89	fpS_ISSOCK	188
1.4.90	fpTime	188
1.4.91	FpTimes	189
1.4.92	FpUmask	189
1.4.93	FpUname	190
1.4.94	FpUnlink	190
1.4.95	FpUtime	190
1.4.96	FpWait	191
1.4.97	FpWaitPid	192
1.4.98	FpWrite	192
1.4.99	FpWriteV	193
1.4.100	FreeShellArgV	193
1.4.101	wexitStatus	193
1.4.102	wifexited	193

1.4.103	wifsignaled	194
1.4.104	wstopsig	194
1.4.105	wtermSIG	194
2	Reference for unit 'Classes'	195
2.1	Used units	195
2.2	Overview	195
2.3	Constants, types and variables	195
2.3.1	Constants	195
2.3.2	Types	198
2.3.3	Variables	211
2.4	Procedures and functions	212
2.4.1	ActivateClassGroup	212
2.4.2	BeginGlobalLoading	212
2.4.3	BinToHex	212
2.4.4	Bounds	213
2.4.5	CheckSynchronize	213
2.4.6	ClassGroupOf	213
2.4.7	CollectionsEqual	214
2.4.8	EndGlobalLoading	214
2.4.9	ExtractStrings	214
2.4.10	FindClass	214
2.4.11	FindGlobalComponent	215
2.4.12	FindIdentToInt	215
2.4.13	FindIntToIdent	215
2.4.14	FindNestedComponent	215
2.4.15	GetClass	216
2.4.16	GetFixupInstanceNames	216
2.4.17	GetFixupReferenceNames	216
2.4.18	GlobalFixupReferences	216
2.4.19	GroupDescendentsWith	217
2.4.20	HexToBin	217
2.4.21	IdentToInt	217
2.4.22	InitComponentRes	217
2.4.23	InitInheritedComponent	218
2.4.24	IntToIdent	218
2.4.25	InvalidPoint	218
2.4.26	LineStart	218
2.4.27	NotifyGlobalLoading	219
2.4.28	ObjectBinaryToText	219

2.4.29	ObjectResourceToText	219
2.4.30	ObjectTextToBinary	219
2.4.31	ObjectTextToResource	219
2.4.32	Point	220
2.4.33	PointsEqual	220
2.4.34	ReadComponentRes	220
2.4.35	ReadComponentResEx	221
2.4.36	ReadComponentResFile	221
2.4.37	Rect	221
2.4.38	RedirectFixupReferences	221
2.4.39	RegisterClass	222
2.4.40	RegisterClassAlias	222
2.4.41	RegisterClasses	222
2.4.42	RegisterComponents	222
2.4.43	RegisterFindGlobalComponentProc	223
2.4.44	RegisterInitComponentHandler	223
2.4.45	RegisterIntegerConsts	223
2.4.46	RegisterNoIcon	224
2.4.47	RegisterNonActiveX	224
2.4.48	RemoveFixupReferences	224
2.4.49	RemoveFixups	224
2.4.50	SmallPoint	225
2.4.51	StartClassGroup	225
2.4.52	UnRegisterClass	225
2.4.53	UnRegisterClasses	225
2.4.54	UnregisterFindGlobalComponentProc	226
2.4.55	UnRegisterModuleClasses	226
2.4.56	WriteComponentResFile	226
2.5	EBitsError	226
2.5.1	Description	226
2.6	EClassNotFound	227
2.6.1	Description	227
2.7	EComponentError	227
2.7.1	Description	227
2.8	EFCREATEError	227
2.8.1	Description	227
2.9	EFilerError	227
2.9.1	Description	227
2.10	EFOpenError	227
2.10.1	Description	227

2.11	EInvalidImage	228
2.11.1	Description	228
2.12	EInvalidOperation	228
2.12.1	Description	228
2.13	EListError	228
2.13.1	Description	228
2.14	EMethodNotFound	228
2.14.1	Description	228
2.15	EObserver	228
2.15.1	Description	228
2.16	EOutOfResources	229
2.16.1	Description	229
2.17	EParseError	229
2.17.1	Description	229
2.18	EReadError	229
2.18.1	Description	229
2.19	EResNotFound	229
2.19.1	Description	229
2.20	EStreamError	229
2.20.1	Description	229
2.21	EStringListError	230
2.21.1	Description	230
2.22	EThread	230
2.22.1	Description	230
2.23	EThreadDestroyCalled	230
2.23.1	Description	230
2.24	EThreadExternalException	230
2.24.1	Description	230
2.25	EWriteError	230
2.25.1	Description	230
2.26	IDesignerNotify	231
2.26.1	Description	231
2.26.2	Method overview	231
2.26.3	IDesignerNotify.Modified	231
2.26.4	IDesignerNotify.Notification	231
2.27	IFPObserved	231
2.27.1	Description	231
2.27.2	Method overview	232
2.27.3	IFPObserved.FPOAttachObserver	232
2.27.4	IFPObserved.FPODetachObserver	232

2.27.5	IFPObserved.FPONotifyObservers	232
2.28	IFPObserver	233
2.28.1	Description	233
2.28.2	Method overview	233
2.28.3	IFPObserver.FPObservedChanged	233
2.29	IInterfaceComponentReference	233
2.29.1	Description	233
2.29.2	Method overview	234
2.29.3	IInterfaceComponentReference.GetComponent	234
2.30	IInterfaceList	234
2.30.1	Description	234
2.30.2	Method overview	234
2.30.3	Property overview	235
2.30.4	IInterfaceList.Get	235
2.30.5	IInterfaceList.GetCapacity	235
2.30.6	IInterfaceList.GetCount	235
2.30.7	IInterfaceList.Put	235
2.30.8	IInterfaceList.SetCapacity	236
2.30.9	IInterfaceList.SetCount	236
2.30.10	IInterfaceList.Clear	236
2.30.11	IInterfaceList.Delete	236
2.30.12	IInterfaceList.Exchange	237
2.30.13	IInterfaceList.First	237
2.30.14	IInterfaceList.IndexOf	237
2.30.15	IInterfaceList.Add	237
2.30.16	IInterfaceList.Insert	237
2.30.17	IInterfaceList.Last	238
2.30.18	IInterfaceList.Remove	238
2.30.19	IInterfaceList.Lock	238
2.30.20	IInterfaceList.Unlock	238
2.30.21	IInterfaceList.Capacity	238
2.30.22	IInterfaceList.Count	239
2.30.23	IInterfaceList.Items	239
2.31	IStreamPersist	239
2.31.1	Description	239
2.31.2	Method overview	239
2.31.3	IStreamPersist.LoadFromStream	240
2.31.4	IStreamPersist.SaveToStream	240
2.32	IStringsAdapter	240
2.32.1	Description	240

2.32.2	Method overview	240
2.32.3	IStringsAdapter.ReferenceStrings	240
2.32.4	IStringsAdapter.ReleaseStrings	240
2.33	IVCLComObject	241
2.33.1	Description	241
2.33.2	Method overview	241
2.33.3	IVCLComObject.GetTypeInfoCount	241
2.33.4	IVCLComObject.GetTypeInfo	241
2.33.5	IVCLComObject.GetIDsOfNames	242
2.33.6	IVCLComObject.Invoke	242
2.33.7	IVCLComObject.SafeCallException	242
2.33.8	IVCLComObject.FreeOnRelease	242
2.34	TAbstractObjectReader	243
2.34.1	Description	243
2.34.2	Method overview	243
2.34.3	TAbstractObjectReader.NextValue	243
2.34.4	TAbstractObjectReader.ReadValue	244
2.34.5	TAbstractObjectReader.BeginRootComponent	244
2.34.6	TAbstractObjectReader.BeginComponent	244
2.34.7	TAbstractObjectReader.BeginProperty	245
2.34.8	TAbstractObjectReader.Read	245
2.34.9	TAbstractObjectReader.ReadBinary	245
2.34.10	TAbstractObjectReader.ReadFloat	245
2.34.11	TAbstractObjectReader.ReadSingle	246
2.34.12	TAbstractObjectReader.ReadDate	246
2.34.13	TAbstractObjectReader.ReadCurrency	246
2.34.14	TAbstractObjectReader.ReadIdent	247
2.34.15	TAbstractObjectReader.ReadInt8	247
2.34.16	TAbstractObjectReader.ReadInt16	247
2.34.17	TAbstractObjectReader.ReadInt32	248
2.34.18	TAbstractObjectReader.ReadInt64	248
2.34.19	TAbstractObjectReader.ReadSet	248
2.34.20	TAbstractObjectReader.ReadStr	249
2.34.21	TAbstractObjectReader.ReadString	249
2.34.22	TAbstractObjectReader.ReadWideString	249
2.34.23	TAbstractObjectReader.ReadUnicodeString	250
2.34.24	TAbstractObjectReader.SkipComponent	250
2.34.25	TAbstractObjectReader.SkipValue	250
2.35	TAbstractObjectWriter	250
2.35.1	Description	250

2.35.2	Method overview	251
2.35.3	TAbstractObjectWriter.BeginCollection	251
2.35.4	TAbstractObjectWriter.BeginComponent	251
2.35.5	TAbstractObjectWriter.BeginList	251
2.35.6	TAbstractObjectWriter.EndList	252
2.35.7	TAbstractObjectWriter.BeginProperty	252
2.35.8	TAbstractObjectWriter.EndProperty	252
2.35.9	TAbstractObjectWriter.Write	252
2.35.10	TAbstractObjectWriter.WriteBinary	252
2.35.11	TAbstractObjectWriter.WriteBoolean	253
2.35.12	TAbstractObjectWriter.WriteFloat	253
2.35.13	TAbstractObjectWriter.WriteSingle	253
2.35.14	TAbstractObjectWriter.WriteDate	253
2.35.15	TAbstractObjectWriter.WriteCurrency	253
2.35.16	TAbstractObjectWriter.WriteIdent	254
2.35.17	TAbstractObjectWriter.WriteInteger	254
2.35.18	TAbstractObjectWriter.WriteUInt64	254
2.35.19	TAbstractObjectWriter.WriteVariant	254
2.35.20	TAbstractObjectWriter.WriteMethodName	254
2.35.21	TAbstractObjectWriter.WriteSet	255
2.35.22	TAbstractObjectWriter.WriteString	255
2.35.23	TAbstractObjectWriter.WriteWideString	255
2.35.24	TAbstractObjectWriter.WriteUnicodeString	255
2.36	TBasicAction	255
2.36.1	Description	255
2.36.2	Method overview	256
2.36.3	Property overview	256
2.36.4	TBasicAction.Create	256
2.36.5	TBasicAction.Destroy	256
2.36.6	TBasicAction.HandlesTarget	256
2.36.7	TBasicAction.UpdateTarget	257
2.36.8	TBasicAction.ExecuteTarget	257
2.36.9	TBasicAction.Execute	257
2.36.10	TBasicAction.RegisterChanges	258
2.36.11	TBasicAction.UnRegisterChanges	258
2.36.12	TBasicAction.Update	258
2.36.13	TBasicAction.ActionComponent	258
2.36.14	TBasicAction.OnExecute	259
2.36.15	TBasicAction.OnUpdate	259
2.37	TBasicActionLink	259

2.37.1	Description	259
2.37.2	Method overview	259
2.37.3	Property overview	260
2.37.4	TBasicActionLink.Create	260
2.37.5	TBasicActionLink.Destroy	260
2.37.6	TBasicActionLink.Execute	260
2.37.7	TBasicActionLink.Update	261
2.37.8	TBasicActionLink.Action	261
2.37.9	TBasicActionLink.OnChange	261
2.38	TBinaryObjectReader	261
2.38.1	Description	261
2.38.2	Method overview	262
2.38.3	TBinaryObjectReader.Create	262
2.38.4	TBinaryObjectReader.Destroy	262
2.38.5	TBinaryObjectReader.NextValue	263
2.38.6	TBinaryObjectReader.ReadValue	263
2.38.7	TBinaryObjectReader.BeginRootComponent	263
2.38.8	TBinaryObjectReader.BeginComponent	263
2.38.9	TBinaryObjectReader.BeginProperty	263
2.38.10	TBinaryObjectReader.Read	264
2.38.11	TBinaryObjectReader.ReadBinary	264
2.38.12	TBinaryObjectReader.ReadFloat	264
2.38.13	TBinaryObjectReader.ReadSingle	264
2.38.14	TBinaryObjectReader.ReadDate	264
2.38.15	TBinaryObjectReader.ReadCurrency	265
2.38.16	TBinaryObjectReader.ReadIdent	265
2.38.17	TBinaryObjectReader.ReadInt8	265
2.38.18	TBinaryObjectReader.ReadInt16	265
2.38.19	TBinaryObjectReader.ReadInt32	266
2.38.20	TBinaryObjectReader.ReadInt64	266
2.38.21	TBinaryObjectReader.ReadSet	266
2.38.22	TBinaryObjectReader.ReadStr	266
2.38.23	TBinaryObjectReader.ReadString	266
2.38.24	TBinaryObjectReader.ReadWideString	267
2.38.25	TBinaryObjectReader.ReadUnicodeString	267
2.38.26	TBinaryObjectReader.SkipComponent	267
2.38.27	TBinaryObjectReader.SkipValue	267
2.39	TBinaryObjectWriter	268
2.39.1	Description	268
2.39.2	Method overview	268

2.39.3	TBinaryObjectWriter.Create	268
2.39.4	TBinaryObjectWriter.Destroy	268
2.39.5	TBinaryObjectWriter.BeginCollection	269
2.39.6	TBinaryObjectWriter.BeginComponent	269
2.39.7	TBinaryObjectWriter.BeginList	269
2.39.8	TBinaryObjectWriter.EndList	269
2.39.9	TBinaryObjectWriter.BeginProperty	269
2.39.10	TBinaryObjectWriter.EndProperty	269
2.39.11	TBinaryObjectWriter.Write	270
2.39.12	TBinaryObjectWriter.WriteBinary	270
2.39.13	TBinaryObjectWriter.WriteBoolean	270
2.39.14	TBinaryObjectWriter.WriteFloat	270
2.39.15	TBinaryObjectWriter.WriteSingle	270
2.39.16	TBinaryObjectWriter.WriteDate	270
2.39.17	TBinaryObjectWriter.WriteCurrency	271
2.39.18	TBinaryObjectWriter.WriteIdent	271
2.39.19	TBinaryObjectWriter.WriteInteger	271
2.39.20	TBinaryObjectWriter.WriteUInt64	271
2.39.21	TBinaryObjectWriter.WriteMethodName	271
2.39.22	TBinaryObjectWriter.WriteSet	271
2.39.23	TBinaryObjectWriter.WriteString	272
2.39.24	TBinaryObjectWriter.WriteString	272
2.39.25	TBinaryObjectWriter.WriteWideString	272
2.39.26	TBinaryObjectWriter.WriteUnicodeString	272
2.39.27	TBinaryObjectWriter.WriteVariant	272
2.40	TBits	273
2.40.1	Description	273
2.40.2	Method overview	273
2.40.3	Property overview	273
2.40.4	TBits.Create	273
2.40.5	TBits.Destroy	274
2.40.6	TBits.GetFSize	274
2.40.7	TBits.SetOn	274
2.40.8	TBits.Clear	274
2.40.9	TBits.Clearall	275
2.40.10	TBits.AndBits	275
2.40.11	TBits.OrBits	275
2.40.12	TBits.XorBits	275
2.40.13	TBits.NotBits	276
2.40.14	TBits.Get	276

2.40.15	TBits.Grow	276
2.40.16	TBits.Equals	277
2.40.17	TBits.SetIndex	277
2.40.18	TBits.FindFirstBit	277
2.40.19	TBits.FindNextBit	278
2.40.20	TBits.FindPrevBit	278
2.40.21	TBits.OpenBit	278
2.40.22	TBits.Bits	279
2.40.23	TBits.Size	279
2.41	TBytesStream	279
2.41.1	Description	279
2.41.2	Method overview	279
2.41.3	Property overview	279
2.41.4	TBytesStream.Create	280
2.41.5	TBytesStream.Bytes	280
2.42	TCollection	280
2.42.1	Description	280
2.42.2	Method overview	281
2.42.3	Property overview	281
2.42.4	TCollection.Create	281
2.42.5	TCollection.Destroy	281
2.42.6	TCollection.Owner	282
2.42.7	TCollection.Add	282
2.42.8	TCollection.Assign	282
2.42.9	TCollection.BeginUpdate	282
2.42.10	TCollection.Clear	283
2.42.11	TCollection.EndUpdate	283
2.42.12	TCollection.Delete	283
2.42.13	TCollection.GetEnumerator	284
2.42.14	TCollection.GetNamePath	284
2.42.15	TCollection.Insert	284
2.42.16	TCollection.FindItemID	284
2.42.17	TCollection.Exchange	285
2.42.18	TCollection.Sort	285
2.42.19	TCollection.Count	285
2.42.20	TCollection.ItemClass	286
2.42.21	TCollection.Items	286
2.43	TCollectionEnumerator	286
2.43.1	Description	286
2.43.2	Method overview	286

2.43.3	Property overview	287
2.43.4	TCollectionEnumerator.Create	287
2.43.5	TCollectionEnumerator.GetCurrent	287
2.43.6	TCollectionEnumerator.MoveNext	287
2.43.7	TCollectionEnumerator.Current	287
2.44	TCollectionItem	288
2.44.1	Description	288
2.44.2	Method overview	288
2.44.3	Property overview	288
2.44.4	TCollectionItem.Create	288
2.44.5	TCollectionItem.Destroy	288
2.44.6	TCollectionItem.GetNamePath	289
2.44.7	TCollectionItem.Collection	289
2.44.8	TCollectionItem.ID	289
2.44.9	TCollectionItem.Index	290
2.44.10	TCollectionItem.DisplayName	290
2.45	TComponent	290
2.45.1	Description	290
2.45.2	Interfaces overview	291
2.45.3	Method overview	291
2.45.4	Property overview	291
2.45.5	TComponent.Notification	292
2.45.6	TComponent.WriteState	292
2.45.7	TComponent.Create	292
2.45.8	TComponent.Destroy	293
2.45.9	TComponent.BeforeDestruction	293
2.45.10	TComponent.DestroyComponents	293
2.45.11	TComponent.Destroying	293
2.45.12	TComponent.ExecuteAction	294
2.45.13	TComponent.FindComponent	294
2.45.14	TComponent.FreeNotification	294
2.45.15	TComponent.RemoveFreeNotification	294
2.45.16	TComponent.FreeOnRelease	295
2.45.17	TComponent.GetEnumerator	295
2.45.18	TComponent.GetNamePath	295
2.45.19	TComponent.GetParentComponent	295
2.45.20	TComponent.HasParent	296
2.45.21	TComponent.InsertComponent	296
2.45.22	TComponent.RemoveComponent	296
2.45.23	TComponent.SafeCallException	296

2.45.24	TComponent.SetSubComponent	297
2.45.25	TComponent.UpdateAction	297
2.45.26	TComponent.IsImplementorOf	297
2.45.27	TComponent.ReferenceInterface	297
2.45.28	TComponent.ComObject	298
2.45.29	TComponent.Components	298
2.45.30	TComponent.ComponentCount	298
2.45.31	TComponent.ComponentIndex	298
2.45.32	TComponent.ComponentState	299
2.45.33	TComponent.ComponentStyle	299
2.45.34	TComponent.DesignInfo	299
2.45.35	TComponent.Owner	300
2.45.36	TComponent.VCLComObject	300
2.45.37	TComponent.Name	300
2.45.38	TComponent.Tag	300
2.46	TComponentEnumerator	301
2.46.1	Description	301
2.46.2	Method overview	301
2.46.3	Property overview	301
2.46.4	TComponentEnumerator.Create	301
2.46.5	TComponentEnumerator.GetCurrent	301
2.46.6	TComponentEnumerator.MoveNext	302
2.46.7	TComponentEnumerator.Current	302
2.47	TCustomMemoryStream	302
2.47.1	Description	302
2.47.2	Method overview	302
2.47.3	Property overview	303
2.47.4	TCustomMemoryStream.Read	303
2.47.5	TCustomMemoryStream.Seek	303
2.47.6	TCustomMemoryStream.SaveToStream	303
2.47.7	TCustomMemoryStream.SaveToFile	304
2.47.8	TCustomMemoryStream.Memory	304
2.48	TDataModule	304
2.48.1	Description	304
2.48.2	Method overview	305
2.48.3	Property overview	305
2.48.4	TDataModule.Create	305
2.48.5	TDataModule.CreateNew	305
2.48.6	TDataModule.Destroy	305
2.48.7	TDataModule.AfterConstruction	306

2.48.8	TDataModule.BeforeDestruction	306
2.48.9	TDataModule.DesignOffset	306
2.48.10	TDataModule.DesignSize	307
2.48.11	TDataModule.OnCreate	307
2.48.12	TDataModule.OnDestroy	307
2.48.13	TDataModule.OldCreateOrder	307
2.49	TFile	308
2.49.1	Description	308
2.49.2	Method overview	308
2.49.3	Property overview	308
2.49.4	TFile.DefineProperty	308
2.49.5	TFile.DefineBinaryProperty	308
2.49.6	TFile.Root	309
2.49.7	TFile.LookupRoot	309
2.49.8	TFile.Ancestor	309
2.49.9	TFile.IgnoreChildren	309
2.50	TFileStream	310
2.50.1	Description	310
2.50.2	Method overview	310
2.50.3	Property overview	310
2.50.4	TFileStream.Create	310
2.50.5	TFileStream.Destroy	310
2.50.6	TFileStream.FileName	311
2.51	TFPList	311
2.51.1	Description	311
2.51.2	Method overview	312
2.51.3	Property overview	312
2.51.4	TFPList.Destroy	312
2.51.5	TFPList.AddList	312
2.51.6	TFPList.Add	313
2.51.7	TFPList.Clear	313
2.51.8	TFPList.Delete	313
2.51.9	TFPList.Error	313
2.51.10	TFPList.Exchange	314
2.51.11	TFPList.Expand	314
2.51.12	TFPList.Extract	314
2.51.13	TFPList.First	314
2.51.14	TFPList.GetEnumerator	315
2.51.15	TFPList.IndexOf	315
2.51.16	TFPList.IndexOfItem	315

2.51.17	TFPList.Insert	315
2.51.18	TFPList.Last	316
2.51.19	TFPList.Move	316
2.51.20	TFPList.Assign	316
2.51.21	TFPList.Remove	316
2.51.22	TFPList.Pack	317
2.51.23	TFPList.Sort	317
2.51.24	TFPList.ForEachCall	317
2.51.25	TFPList.Capacity	318
2.51.26	TFPList.Count	318
2.51.27	TFPList.Items	318
2.51.28	TFPList.List	318
2.52	TFPListEnumerator	319
2.52.1	Description	319
2.52.2	Method overview	319
2.52.3	Property overview	319
2.52.4	TFPListEnumerator.Create	319
2.52.5	TFPListEnumerator.GetCurrent	319
2.52.6	TFPListEnumerator.MoveNext	319
2.52.7	TFPListEnumerator.Current	320
2.53	THandleStream	320
2.53.1	Description	320
2.53.2	Method overview	320
2.53.3	Property overview	320
2.53.4	THandleStream.Create	321
2.53.5	THandleStream.Read	321
2.53.6	THandleStream.Write	321
2.53.7	THandleStream.Seek	321
2.53.8	THandleStream.Handle	322
2.54	TInterfacedPersistent	322
2.54.1	Description	322
2.54.2	Interfaces overview	322
2.54.3	Method overview	322
2.54.4	TInterfacedPersistent.QueryInterface	322
2.54.5	TInterfacedPersistent.AfterConstruction	323
2.55	TInterfaceList	323
2.55.1	Description	323
2.55.2	Interfaces overview	323
2.55.3	Method overview	323
2.55.4	Property overview	323

2.55.5	TInterfaceList.Create	324
2.55.6	TInterfaceList.Destroy	324
2.55.7	TInterfaceList.Clear	324
2.55.8	TInterfaceList.Delete	324
2.55.9	TInterfaceList.Exchange	325
2.55.10	TInterfaceList.First	325
2.55.11	TInterfaceList.GetEnumerator	325
2.55.12	TInterfaceList.IndexOf	325
2.55.13	TInterfaceList.Add	326
2.55.14	TInterfaceList.Insert	326
2.55.15	TInterfaceList.Last	326
2.55.16	TInterfaceList.Remove	326
2.55.17	TInterfaceList.Lock	327
2.55.18	TInterfaceList.Unlock	327
2.55.19	TInterfaceList.Expand	327
2.55.20	TInterfaceList.Capacity	327
2.55.21	TInterfaceList.Count	328
2.55.22	TInterfaceList.Items	328
2.56	TInterfaceListEnumerator	328
2.56.1	Description	328
2.56.2	Method overview	328
2.56.3	Property overview	328
2.56.4	TInterfaceListEnumerator.Create	329
2.56.5	TInterfaceListEnumerator.GetCurrent	329
2.56.6	TInterfaceListEnumerator.MoveNext	329
2.56.7	TInterfaceListEnumerator.Current	329
2.57	TList	330
2.57.1	Description	330
2.57.2	Interfaces overview	330
2.57.3	Method overview	330
2.57.4	Property overview	331
2.57.5	TList.Create	331
2.57.6	TList.Destroy	331
2.57.7	TList.FPOAttachObserver	331
2.57.8	TList.FPODetachObserver	331
2.57.9	TList.FPONotifyObservers	332
2.57.10	TList.AddList	332
2.57.11	TList.Add	332
2.57.12	TList.Clear	333
2.57.13	TList.Delete	333

2.57.14	TList.Error	333
2.57.15	TList.Exchange	333
2.57.16	TList.Expand	333
2.57.17	TList.Extract	334
2.57.18	TList.First	334
2.57.19	TList.GetEnumerator	334
2.57.20	TList.IndexOf	335
2.57.21	TList.Insert	335
2.57.22	TList.Last	335
2.57.23	TList.Move	335
2.57.24	TList.Assign	336
2.57.25	TList.Remove	336
2.57.26	TList.Pack	336
2.57.27	TList.Sort	336
2.57.28	TList.Capacity	337
2.57.29	TList.Count	337
2.57.30	TList.Items	337
2.57.31	TList.List	338
2.58	TListEnumerator	338
2.58.1	Description	338
2.58.2	Method overview	338
2.58.3	Property overview	338
2.58.4	TListEnumerator.Create	338
2.58.5	TListEnumerator.GetCurrent	338
2.58.6	TListEnumerator.MoveNext	339
2.58.7	TListEnumerator.Current	339
2.59	TMemoryStream	339
2.59.1	Description	339
2.59.2	Method overview	340
2.59.3	TMemoryStream.Destroy	340
2.59.4	TMemoryStream.Clear	340
2.59.5	TMemoryStream.LoadFromStream	340
2.59.6	TMemoryStream.LoadFromFile	341
2.59.7	TMemoryStream.SetSize	341
2.59.8	TMemoryStream.Write	341
2.60	TOwnedCollection	341
2.60.1	Description	341
2.60.2	Method overview	342
2.60.3	TOwnedCollection.Create	342
2.61	TOwnerStream	342

2.61.1	Description	342
2.61.2	Method overview	342
2.61.3	Property overview	342
2.61.4	TOwnerStream.Create	342
2.61.5	TOwnerStream.Destroy	343
2.61.6	TOwnerStream.Source	343
2.61.7	TOwnerStream.SourceOwner	343
2.62	TParser	343
2.62.1	Description	343
2.62.2	Method overview	344
2.62.3	Property overview	344
2.62.4	TParser.Create	344
2.62.5	TParser.Destroy	344
2.62.6	TParser.CheckToken	345
2.62.7	TParser.CheckTokenSymbol	345
2.62.8	TParser.Error	345
2.62.9	TParser.ErrorFmt	345
2.62.10	TParser.ErrorStr	345
2.62.11	TParser.HexToBinary	346
2.62.12	TParser.NextToken	346
2.62.13	TParser.SourcePos	346
2.62.14	TParser.TokenComponentIdent	347
2.62.15	TParser.TokenFloat	347
2.62.16	TParser.TokenInt	347
2.62.17	TParser.TokenString	348
2.62.18	TParser.TokenWideString	348
2.62.19	TParser.TokenSymbolIs	348
2.62.20	TParser.FloatType	349
2.62.21	TParser.SourceLine	349
2.62.22	TParser.Token	349
2.63	TPersistent	350
2.63.1	Description	350
2.63.2	Interfaces overview	350
2.63.3	Method overview	350
2.63.4	TPersistent.Destroy	350
2.63.5	TPersistent.Assign	351
2.63.6	TPersistent.FPOAttachObserver	351
2.63.7	TPersistent.FPODetachObserver	352
2.63.8	TPersistent.FPONotifyObservers	352
2.63.9	TPersistent.GetNamePath	352

2.64	TProxyStream	352
2.64.1	Description	352
2.64.2	Method overview	353
2.64.3	TProxyStream.Create	353
2.64.4	TProxyStream.Read	353
2.64.5	TProxyStream.Write	353
2.64.6	TProxyStream.Seek	353
2.64.7	TProxyStream.Check	353
2.65	TReader	354
2.65.1	Description	354
2.65.2	Method overview	355
2.65.3	Property overview	356
2.65.4	TReader.Create	356
2.65.5	TReader.Destroy	356
2.65.6	TReader.BeginReferences	356
2.65.7	TReader.CheckValue	357
2.65.8	TReader.DefineProperty	357
2.65.9	TReader.DefineBinaryProperty	357
2.65.10	TReader.EndOfList	357
2.65.11	TReader.EndReferences	357
2.65.12	TReader.FixupReferences	358
2.65.13	TReader.NextValue	358
2.65.14	TReader.Read	358
2.65.15	TReader.ReadBoolean	358
2.65.16	TReader.ReadChar	358
2.65.17	TReader.ReadWideChar	359
2.65.18	TReader.ReadUnicodeChar	359
2.65.19	TReader.ReadCollection	359
2.65.20	TReader.ReadComponent	359
2.65.21	TReader.ReadComponents	359
2.65.22	TReader.ReadFloat	360
2.65.23	TReader.ReadSingle	360
2.65.24	TReader.ReadDate	360
2.65.25	TReader.ReadCurrency	360
2.65.26	TReader.ReadIdent	360
2.65.27	TReader.ReadInteger	361
2.65.28	TReader.ReadInt64	361
2.65.29	TReader.ReadSet	361
2.65.30	TReader.ReadListBegin	361
2.65.31	TReader.ReadListEnd	361

2.65.32	TReader.ReadRootComponent	362
2.65.33	TReader.ReadVariant	362
2.65.34	TReader.ReadString	362
2.65.35	TReader.ReadWideString	362
2.65.36	TReader.ReadUnicodeString	362
2.65.37	TReader.ReadValue	363
2.65.38	TReader.CopyValue	363
2.65.39	TReader.Driver	363
2.65.40	TReader.Owner	363
2.65.41	TReader.Parent	363
2.65.42	TReader.OnError	364
2.65.43	TReader.OnPropertyNotFound	364
2.65.44	TReader.OnFindMethod	364
2.65.45	TReader.OnSetMethodProperty	364
2.65.46	TReader.OnSetName	365
2.65.47	TReader.OnReferenceName	365
2.65.48	TReader.OnAncestorNotFound	365
2.65.49	TReader.OnCreateComponent	365
2.65.50	TReader.OnFindComponentClass	365
2.65.51	TReader.OnReadStringProperty	366
2.66	TRecall	366
2.66.1	Description	366
2.66.2	Method overview	366
2.66.3	Property overview	366
2.66.4	TRecall.Create	367
2.66.5	TRecall.Destroy	367
2.66.6	TRecall.Store	367
2.66.7	TRecall.Forget	367
2.66.8	TRecall.Reference	368
2.67	TResourceStream	368
2.67.1	Description	368
2.67.2	Method overview	368
2.67.3	TResourceStream.Create	368
2.67.4	TResourceStream.CreateFromID	368
2.67.5	TResourceStream.Destroy	369
2.68	TStream	369
2.68.1	Description	369
2.68.2	Method overview	369
2.68.3	Property overview	370
2.68.4	TStream.Read	370

2.68.5	TStream.Write	370
2.68.6	TStream.Seek	371
2.68.7	TStream.ReadBuffer	371
2.68.8	TStream.WriteBuffer	372
2.68.9	TStream.CopyFrom	372
2.68.10	TStream.ReadComponent	372
2.68.11	TStream.ReadComponentRes	373
2.68.12	TStream.WriteComponent	373
2.68.13	TStream.WriteComponentRes	373
2.68.14	TStream.WriteDescendent	374
2.68.15	TStream.WriteDescendentRes	374
2.68.16	TStream.WriteResourceHeader	374
2.68.17	TStream.FixupResourceHeader	374
2.68.18	TStream.ReadResHeader	375
2.68.19	TStream.ReadByte	375
2.68.20	TStream.ReadWord	375
2.68.21	TStream.ReadDWord	376
2.68.22	TStream.ReadQWord	376
2.68.23	TStream.ReadAnsiString	376
2.68.24	TStream.WriteByte	376
2.68.25	TStream.WriteWord	377
2.68.26	TStream.WriteDWord	377
2.68.27	TStream.WriteQWord	377
2.68.28	TStream.WriteAnsiString	378
2.68.29	TStream.Position	378
2.68.30	TStream.Size	378
2.69	TStreamAdapter	379
2.69.1	Description	379
2.69.2	Interfaces overview	379
2.69.3	Method overview	379
2.69.4	Property overview	379
2.69.5	TStreamAdapter.Create	379
2.69.6	TStreamAdapter.Destroy	379
2.69.7	TStreamAdapter.Read	380
2.69.8	TStreamAdapter.Write	380
2.69.9	TStreamAdapter.Seek	380
2.69.10	TStreamAdapter.SetSize	381
2.69.11	TStreamAdapter.CopyTo	381
2.69.12	TStreamAdapter.Commit	381
2.69.13	TStreamAdapter.Revert	381

2.69.14	TStreamAdapter.LockRegion	382
2.69.15	TStreamAdapter.UnlockRegion	382
2.69.16	TStreamAdapter.Stat	382
2.69.17	TStreamAdapter.Clone	382
2.69.18	TStreamAdapter.Stream	383
2.69.19	TStreamAdapter.StreamOwnership	383
2.70	TStringList	383
2.70.1	Description	383
2.70.2	Method overview	383
2.70.3	Property overview	384
2.70.4	TStringList.Destroy	384
2.70.5	TStringList.Add	384
2.70.6	TStringList.Clear	384
2.70.7	TStringList.Delete	385
2.70.8	TStringList.Exchange	385
2.70.9	TStringList.Find	385
2.70.10	TStringList.IndexOf	385
2.70.11	TStringList.Insert	386
2.70.12	TStringList.Sort	386
2.70.13	TStringList.CustomSort	386
2.70.14	TStringList.Duplicates	386
2.70.15	TStringList.Sorted	387
2.70.16	TStringList.CaseSensitive	387
2.70.17	TStringList.OnChange	387
2.70.18	TStringList.OnChanging	388
2.70.19	TStringList.OwnsObjects	388
2.71	TStrings	388
2.71.1	Description	388
2.71.2	Method overview	389
2.71.3	Property overview	390
2.71.4	TStrings.Destroy	390
2.71.5	TStrings.Add	390
2.71.6	TStrings.AddObject	390
2.71.7	TStrings.Append	391
2.71.8	TStrings.AddStrings	391
2.71.9	TStrings.AddText	391
2.71.10	TStrings.Assign	391
2.71.11	TStrings.BeginUpdate	392
2.71.12	TStrings.Clear	392
2.71.13	TStrings.Delete	392

2.71.14	TStrings.EndUpdate	393
2.71.15	TStrings.Equals	393
2.71.16	TStrings.Exchange	393
2.71.17	TStrings.GetEnumerator	394
2.71.18	TStrings.GetText	394
2.71.19	TStrings.IndexOf	394
2.71.20	TStrings.IndexOfName	394
2.71.21	TStrings.IndexOfObject	395
2.71.22	TStrings.Insert	395
2.71.23	TStrings.InsertObject	395
2.71.24	TStrings.LoadFromFile	396
2.71.25	TStrings.LoadFromStream	396
2.71.26	TStrings.Move	396
2.71.27	TStrings.SaveToFile	397
2.71.28	TStrings.SaveToStream	397
2.71.29	TStrings.SetText	397
2.71.30	TStrings.GetNameValue	398
2.71.31	TStrings.ExtractName	398
2.71.32	TStrings.TextLineBreakStyle	398
2.71.33	TStrings.Delimiter	398
2.71.34	TStrings.DelimitedText	399
2.71.35	TStrings.StrictDelimiter	399
2.71.36	TStrings.QuoteChar	399
2.71.37	TStrings.NameValueSeparator	400
2.71.38	TStrings.ValueFromIndex	400
2.71.39	TStrings.Capacity	400
2.71.40	TStrings.CommaText	400
2.71.41	TStrings.Count	401
2.71.42	TStrings.Names	402
2.71.43	TStrings.Objects	402
2.71.44	TStrings.Values	402
2.71.45	TStrings.Strings	403
2.71.46	TStrings.Text	403
2.71.47	TStrings.StringsAdapter	404
2.72	TStringsEnumerator	404
2.72.1	Description	404
2.72.2	Method overview	404
2.72.3	Property overview	404
2.72.4	TStringsEnumerator.Create	404
2.72.5	TStringsEnumerator.GetCurrent	404

2.72.6	TStringsEnumerator.MoveNext	405
2.72.7	TStringsEnumerator.Current	405
2.73	TStringStream	405
2.73.1	Description	405
2.73.2	Method overview	405
2.73.3	Property overview	406
2.73.4	TStringStream.Create	406
2.73.5	TStringStream.Read	406
2.73.6	TStringStream.ReadString	406
2.73.7	TStringStream.Seek	406
2.73.8	TStringStream.Write	407
2.73.9	TStringStream.WriteString	407
2.73.10	TStringStream.DataString	407
2.74	TTextObjectWriter	407
2.74.1	Description	407
2.75	TThread	407
2.75.1	Description	407
2.75.2	Method overview	408
2.75.3	Property overview	408
2.75.4	TThread.Execute	408
2.75.5	TThread.Synchronize	409
2.75.6	TThread.Queue	409
2.75.7	TThread.Create	409
2.75.8	TThread.Destroy	409
2.75.9	TThread.CreateAnonymousThread	410
2.75.10	TThread.NameThreadForDebugging	410
2.75.11	TThread.SetReturnValue	410
2.75.12	TThread.CheckTerminated	410
2.75.13	TThread.RemoveQueuedEvents	411
2.75.14	TThread.SpinWait	411
2.75.15	TThread.Sleep	411
2.75.16	TThread.Yield	412
2.75.17	TThread.GetSystemTimes	412
2.75.18	TThread.GetTickCount	412
2.75.19	TThread.GetTickCount64	412
2.75.20	TThread.AfterConstruction	412
2.75.21	TThread.Start	413
2.75.22	TThread.Resume	413
2.75.23	TThread.Suspend	413
2.75.24	TThread.Terminate	413

2.75.25	TThread.WaitFor	413
2.75.26	TThread.CurrentThread	414
2.75.27	TThread.ProcessorCount	414
2.75.28	TThread.IsSingleProcessor	414
2.75.29	TThread.FreeOnTerminate	415
2.75.30	TThread.Handle	415
2.75.31	TThread.ExternalThread	415
2.75.32	TThread.Priority	415
2.75.33	TThread.Suspended	415
2.75.34	TThread.Finished	416
2.75.35	TThread.ThreadID	416
2.75.36	TThread.OnTerminate	416
2.75.37	TThread.FatalException	416
2.76	TThreadList	416
2.76.1	Description	416
2.76.2	Method overview	417
2.76.3	Property overview	417
2.76.4	TThreadList.Create	417
2.76.5	TThreadList.Destroy	417
2.76.6	TThreadList.Add	417
2.76.7	TThreadList.Clear	418
2.76.8	TThreadList.LockList	418
2.76.9	TThreadList.Remove	418
2.76.10	TThreadList.UnlockList	418
2.76.11	TThreadList.Duplicates	419
2.77	TWriter	419
2.77.1	Description	419
2.77.2	Method overview	420
2.77.3	Property overview	420
2.77.4	TWriter.Create	420
2.77.5	TWriter.Destroy	421
2.77.6	TWriter.DefineProperty	421
2.77.7	TWriter.DefineBinaryProperty	421
2.77.8	TWriter.Write	421
2.77.9	TWriter.WriteBoolean	421
2.77.10	TWriter.WriteCollection	422
2.77.11	TWriter.WriteComponent	422
2.77.12	TWriter.WriteChar	422
2.77.13	TWriter.WriteWideChar	422
2.77.14	TWriter.WriteDescendent	422

2.77.15	TWriter.WriteFloat	422
2.77.16	TWriter.WriteSingle	423
2.77.17	TWriter.WriteDate	423
2.77.18	TWriter.WriteCurrency	423
2.77.19	TWriter.WriteIdent	423
2.77.20	TWriter.WriteInteger	423
2.77.21	TWriter.WriteSet	424
2.77.22	TWriter.WriteListBegin	424
2.77.23	TWriter.WriteListEnd	424
2.77.24	TWriter.WriteRootComponent	424
2.77.25	TWriter.WriteString	424
2.77.26	TWriter.WriteWideString	425
2.77.27	TWriter.WriteUnicodeString	425
2.77.28	TWriter.WriteVariant	425
2.77.29	TWriter.RootAncestor	425
2.77.30	TWriter.OnFindAncestor	425
2.77.31	TWriter.OnWriteMethodProperty	426
2.77.32	TWriter.OnWriteStringProperty	426
2.77.33	TWriter.Driver	426
2.77.34	TWriter.PropertyPath	426
3	Reference for unit 'clocale'	427
3.1	Overview	427
4	Reference for unit 'cmem'	428
4.1	Overview	428
4.2	Constants, types and variables	428
4.2.1	Constants	428
4.3	Procedures and functions	428
4.3.1	CAlloc	428
4.3.2	Free	428
4.3.3	Malloc	429
4.3.4	ReAlloc	429
5	Reference for unit 'Crt'	430
5.1	Overview	430
5.2	Constants, types and variables	430
5.2.1	Constants	430
5.2.2	Types	433
5.2.3	Variables	433
5.3	Procedures and functions	434

5.3.1	AssignCrt	434
5.3.2	ClrEol	435
5.3.3	ClrScr	435
5.3.4	cursorbig	436
5.3.5	cursoroff	436
5.3.6	cursoron	436
5.3.7	Delay	437
5.3.8	DelLine	437
5.3.9	GotoXY	438
5.3.10	HighVideo	438
5.3.11	InsLine	439
5.3.12	KeyPressed	439
5.3.13	LowVideo	440
5.3.14	NormVideo	440
5.3.15	NoSound	441
5.3.16	ReadKey	441
5.3.17	Sound	442
5.3.18	TextBackground	442
5.3.19	TextColor	443
5.3.20	TextMode	443
5.3.21	WhereX	444
5.3.22	WhereY	444
5.3.23	Window	445
6	Reference for unit 'cthreads'	446
6.1	Overview	446
6.2	Procedures and functions	446
6.2.1	SetCThreadManager	446
7	Reference for unit 'ctypes'	447
7.1	Used units	447
7.2	Overview	447
7.3	Constants, types and variables	447
7.3.1	Types	447
7.4	Procedures and functions	452
7.4.1	operator *(clongdouble, Double): Double	452
7.4.2	operator *(Double, clongdouble): Double	452
7.4.3	operator +(clongdouble, Double): Double	453
7.4.4	operator +(Double, clongdouble): Double	453
7.4.5	operator -(clongdouble, Double): Double	453
7.4.6	operator -(Double, clongdouble): Double	453

7.4.7	operator /(clongdouble, Double): Double	454
7.4.8	operator /(Double, clongdouble): Double	454
7.4.9	operator :=(clongdouble): Double	454
7.4.10	operator :=(Double): clongdouble	454
7.4.11	operator <(clongdouble, Double): Boolean	454
7.4.12	operator <(Double, clongdouble): Boolean	455
7.4.13	operator <=(clongdouble, Double): Boolean	455
7.4.14	operator <=(Double, clongdouble): Boolean	455
7.4.15	operator =(clongdouble, Double): Boolean	455
7.4.16	operator =(Double, clongdouble): Boolean	456
7.4.17	operator >(clongdouble, Double): Boolean	456
7.4.18	operator >(Double, clongdouble): Boolean	456
7.4.19	operator >=(clongdouble, Double): Boolean	456
7.4.20	operator >=(Double, clongdouble): Boolean	457
8	Reference for unit 'cwstring'	458
8.1	Overview	458
8.2	Procedures and functions	458
8.2.1	SetCWidestringManager	458
9	Reference for unit 'dateutils'	459
9.1	Used units	459
9.2	Overview	459
9.3	Constants, types and variables	459
9.3.1	Constants	459
9.4	Procedures and functions	461
9.4.1	CompareDate	461
9.4.2	CompareDateTime	462
9.4.3	CompareTime	463
9.4.4	DateOf	464
9.4.5	DateTimeToDosDateTime	465
9.4.6	DateTimeToJulianDate	465
9.4.7	DateTimeToMac	465
9.4.8	DateTimeToModifiedJulianDate	466
9.4.9	DateTimeToUnix	466
9.4.10	DayOf	466
9.4.11	DayOfTheMonth	466
9.4.12	DayOfTheWeek	467
9.4.13	DayOfTheYear	467
9.4.14	DaysBetween	468
9.4.15	DaysInAMonth	468

9.4.16	DaysInAYear	469
9.4.17	DaysInMonth	470
9.4.18	DaysInYear	470
9.4.19	DaySpan	471
9.4.20	DecodeDateDay	471
9.4.21	DecodeDateMonthWeek	472
9.4.22	DecodeDateTime	473
9.4.23	DecodeDateWeek	473
9.4.24	DecodeDayOfWeekInMonth	474
9.4.25	DosDateTimeToDateTime	474
9.4.26	EncodeDateDay	475
9.4.27	EncodeDateMonthWeek	475
9.4.28	EncodeDateTime	475
9.4.29	EncodeDateWeek	476
9.4.30	EncodeDayOfWeekInMonth	476
9.4.31	EncodeTimeInterval	476
9.4.32	EndOfADay	477
9.4.33	EndOfAMonth	477
9.4.34	EndOfAWeek	478
9.4.35	EndOfAYear	479
9.4.36	EndOfTheDay	479
9.4.37	EndOfTheMonth	480
9.4.38	EndOfTheWeek	480
9.4.39	EndOfTheYear	481
9.4.40	HourOf	481
9.4.41	HourOfTheDay	482
9.4.42	HourOfTheMonth	482
9.4.43	HourOfTheWeek	482
9.4.44	HourOfTheYear	483
9.4.45	HoursBetween	483
9.4.46	HourSpan	484
9.4.47	IncDay	485
9.4.48	IncHour	485
9.4.49	IncMilliSecond	486
9.4.50	IncMinute	486
9.4.51	IncSecond	487
9.4.52	IncWeek	487
9.4.53	IncYear	488
9.4.54	InvalidDateDayError	488
9.4.55	InvalidDateMonthWeekError	489

9.4.56	InvalidDateTimeError	489
9.4.57	InvalidDateWeekError	489
9.4.58	InvalidDayOfWeekInMonthError	490
9.4.59	IsInLeapYear	490
9.4.60	IsPM	491
9.4.61	IsSameDay	491
9.4.62	IsSameMonth	492
9.4.63	IsToday	492
9.4.64	IsValidDate	492
9.4.65	IsValidDateDay	493
9.4.66	IsValidDateMonthWeek	493
9.4.67	IsValidDateTime	494
9.4.68	IsValidDateWeek	495
9.4.69	IsValidTime	496
9.4.70	JulianDateToDateTime	496
9.4.71	LocalTimeToUniversal	497
9.4.72	MacTimeStampToUnix	497
9.4.73	MacToDateTime	497
9.4.74	MilliSecondOf	497
9.4.75	MilliSecondOfTheDay	498
9.4.76	MilliSecondOfTheHour	498
9.4.77	MilliSecondOfTheMinute	498
9.4.78	MilliSecondOfTheMonth	499
9.4.79	MilliSecondOfTheSecond	499
9.4.80	MilliSecondOfTheWeek	499
9.4.81	MilliSecondOfTheYear	500
9.4.82	MilliSecondsBetween	500
9.4.83	MilliSecondSpan	501
9.4.84	MinuteOf	502
9.4.85	MinuteOfTheDay	502
9.4.86	MinuteOfTheHour	502
9.4.87	MinuteOfTheMonth	503
9.4.88	MinuteOfTheWeek	503
9.4.89	MinuteOfTheYear	503
9.4.90	MinutesBetween	504
9.4.91	MinuteSpan	505
9.4.92	ModifiedJulianDateToDateTime	505
9.4.93	MonthOf	506
9.4.94	MonthOfTheYear	506
9.4.95	MonthsBetween	506

9.4.96	MonthSpan	507
9.4.97	NthDayOfWeek	508
9.4.98	PeriodBetween	509
9.4.99	PreviousDayOfWeek	509
9.4.100	RecodeDate	509
9.4.101	RecodeDateTime	510
9.4.102	RecodeDay	511
9.4.103	RecodeHour	511
9.4.104	RecodeMilliSecond	512
9.4.105	RecodeMinute	513
9.4.106	RecodeMonth	513
9.4.107	RecodeSecond	514
9.4.108	RecodeTime	515
9.4.109	RecodeYear	515
9.4.110	SameDate	516
9.4.111	SameDateTime	517
9.4.112	SameTime	518
9.4.113	ScanDateTime	518
9.4.114	SecondOf	519
9.4.115	SecondOfTheDay	519
9.4.116	SecondOfTheHour	520
9.4.117	SecondOfTheMinute	520
9.4.118	SecondOfTheMonth	520
9.4.119	SecondOfTheWeek	521
9.4.120	SecondOfTheYear	521
9.4.121	SecondsBetween	521
9.4.122	SecondSpan	522
9.4.123	StartOfADay	523
9.4.124	StartOfAMonth	524
9.4.125	StartOfAWeek	524
9.4.126	StartOfAYear	525
9.4.127	StartOfTheDay	526
9.4.128	StartOfTheMonth	526
9.4.129	StartOfTheWeek	527
9.4.130	StartOfTheYear	527
9.4.131	TimeOf	528
9.4.132	Today	528
9.4.133	Tomorrow	528
9.4.134	TryEncodeDateDay	529
9.4.135	TryEncodeDateMonthWeek	529

9.4.136	TryEncodeDateTime	530
9.4.137	TryEncodeDateWeek	531
9.4.138	TryEncodeDayOfWeekInMonth	532
9.4.139	TryEncodeTimeInterval	532
9.4.140	TryJulianDateToDateTime	533
9.4.141	TryModifiedJulianDateToDateTime	533
9.4.142	TryRecodeDateTime	533
9.4.143	UniversalTimeToLocal	534
9.4.144	UnixTimeStampToMac	534
9.4.145	UnixToDateTime	535
9.4.146	WeekOf	535
9.4.147	WeekOfTheMonth	535
9.4.148	WeekOfTheYear	536
9.4.149	WeeksBetween	536
9.4.150	WeeksInAYear	537
9.4.151	WeeksInYear	538
9.4.152	WeekSpan	538
9.4.153	WithinPastDays	539
9.4.154	WithinPastHours	540
9.4.155	WithinPastMilliSeconds	541
9.4.156	WithinPastMinutes	542
9.4.157	WithinPastMonths	543
9.4.158	WithinPastSeconds	544
9.4.159	WithinPastWeeks	545
9.4.160	WithinPastYears	546
9.4.161	YearOf	547
9.4.162	YearsBetween	548
9.4.163	YearSpan	549
9.4.164	Yesterday	549
10	Reference for unit 'Dos'	551
10.1	Used units	551
10.2	Overview	551
10.3	System information	551
10.4	Process handling	552
10.5	Directory and disk handling	552
10.6	File handling	553
10.7	File open mode constants.	553
10.8	File attributes	553
10.9	Constants, types and variables	554

10.9.1	Constants	554
10.9.2	Types	555
10.9.3	Variables	557
10.10	Procedures and functions	557
10.10.1	AddDisk	557
10.10.2	DiskFree	558
10.10.3	DiskSize	558
10.10.4	DosExitCode	559
10.10.5	DosVersion	559
10.10.6	DTToUnixDate	560
10.10.7	EnvCount	560
10.10.8	EnvStr	561
10.10.9	Exec	561
10.10.10	FExpand	562
10.10.11	FindClose	562
10.10.12	FindFirst	562
10.10.13	FindNext	563
10.10.14	FSearch	563
10.10.15	FSplit	564
10.10.16	GetCBreak	565
10.10.17	GetDate	565
10.10.18	GetEnv	566
10.10.19	GetFAttr	566
10.10.20	GetFTime	567
10.10.21	GetIntVec	568
10.10.22	GetLongName	568
10.10.23	GetMsCount	568
10.10.24	GetShortName	569
10.10.25	GetTime	569
10.10.26	GetVerify	570
10.10.27	Intr	570
10.10.28	Keep	570
10.10.29	MSDos	571
10.10.30	PackTime	571
10.10.31	SetCBreak	572
10.10.32	SetDate	572
10.10.33	SetFAttr	572
10.10.34	SetFTime	573
10.10.35	SetIntVec	573
10.10.36	SetTime	574

10.10.37	SetVerify	574
10.10.38	SwapVectors	574
10.10.39	UnixDateToDt	575
10.10.40	UnpackTime	575
10.10.41	weekday	575
11	Reference for unit 'dxeload'	576
11.1	Overview	576
11.2	Procedures and functions	576
11.2.1	dxeload	576
12	Reference for unit 'dynlibs'	577
12.1	Overview	577
12.2	Constants, types and variables	577
12.2.1	Constants	577
12.2.2	Types	577
12.3	Procedures and functions	578
12.3.1	FreeLibrary	578
12.3.2	GetLoadErrorStr	578
12.3.3	GetProcAddress	578
12.3.4	GetProcedureAddress	578
12.3.5	LoadLibrary	579
12.3.6	SafeLoadLibrary	579
12.3.7	UnloadLibrary	579
13	Reference for unit 'emu387'	580
13.1	Overview	580
13.2	Procedures and functions	580
13.2.1	npsetup	580
14	Reference for unit 'exeinfo'	581
14.1	Overview	581
14.2	Constants, types and variables	581
14.2.1	Types	581
14.3	Procedures and functions	582
14.3.1	CloseExeFile	582
14.3.2	FindExeSection	582
14.3.3	GetModuleByAddr	582
14.3.4	OpenExeFile	582
14.3.5	ReadDebugLink	583
15	Reference for unit 'getopts'	584

15.1	Overview	584
15.2	Constants, types and variables	584
15.2.1	Constants	584
15.2.2	Types	585
15.2.3	Variables	585
15.3	Procedures and functions	586
15.3.1	GetLongOpts	586
15.3.2	GetOpt	586
16	Reference for unit 'go32'	589
16.1	Overview	589
16.2	Real mode callbacks	589
16.3	Executing software interrupts	590
16.4	Software interrupts	592
16.5	Hardware interrupts	592
16.6	Disabling interrupts	594
16.7	Creating your own interrupt handlers	594
16.8	Protected mode interrupts vs. Real mode interrupts	594
16.9	Handling interrupts with DPMI	594
16.10	Interrupt redirection	595
16.11	Processor access	595
16.12	I/O port access	595
16.13	dos memory access	595
16.14	FPC specialities	595
16.15	Selectors and descriptors	596
16.16	What is DPMI	596
16.17	Constants, types and variables	596
16.17.1	Constants	596
16.17.2	Types	599
16.17.3	Variables	600
16.18	Procedures and functions	600
16.18.1	allocate_ldt_descriptors	600
16.18.2	allocate_memory_block	602
16.18.3	copyfromdos	603
16.18.4	copytodos	603
16.18.5	create_code_segment_alias_descriptor	604
16.18.6	disable	604
16.18.7	dpmi_dosmemfillchar	604
16.18.8	dpmi_dosmemfillword	605
16.18.9	dpmi_dosmemget	605

16.18.10	dpmi_dosmemmove	605
16.18.11	dpmi_dosmempu	605
16.18.12	enable	606
16.18.13	free_ldt_descriptor	606
16.18.14	free_linear_addr_mapping	606
16.18.15	free_memory_block	606
16.18.16	free_rm_callback	607
16.18.17	get_cs	607
16.18.18	get_descriptor_access_right	607
16.18.19	get_dp	608
16.18.20	get_ds	608
16.18.21	get_exception_handler	608
16.18.22	get_linear_addr	609
16.18.23	get_meminfo	609
16.18.24	get_next_selector_increment_value	610
16.18.25	get_page_attributes	611
16.18.26	get_page_size	611
16.18.27	get_pm_exception_handler	611
16.18.28	get_pm_interrupt	611
16.18.29	get_rm_callback	612
16.18.30	get_rm_interrupt	615
16.18.31	get_run_mode	615
16.18.32	get_segment_base_address	616
16.18.33	get_segment_limit	616
16.18.34	get_ss	617
16.18.35	global_dos_alloc	617
16.18.36	global_dos_free	618
16.18.37	inportb	619
16.18.38	inportl	619
16.18.39	inportw	619
16.18.40	lock_code	620
16.18.41	lock_data	620
16.18.42	lock_linear_region	621
16.18.43	map_device_in_memory_block	621
16.18.44	outportb	621
16.18.45	outportl	622
16.18.46	outportw	622
16.18.47	realintr	623
16.18.48	request_linear_region	623
16.18.49	segment_to_descriptor	624

16.18.50	seg_fillchar	624
16.18.51	seg_fillword	625
16.18.52	seg_move	625
16.18.53	set_descriptor_access_right	626
16.18.54	set_exception_handler	626
16.18.55	set_page_attributes	626
16.18.56	set_pm_exception_handler	627
16.18.57	set_pm_interrupt	627
16.18.58	set_rm_interrupt	628
16.18.59	set_segment_base_address	628
16.18.60	set_segment_limit	629
16.18.61	tb_offset	629
16.18.62	tb_segment	629
16.18.63	tb_size	630
16.18.64	transfer_buffer	630
16.18.65	unlock_code	630
16.18.66	unlock_data	631
16.18.67	unlock_linear_region	631
17	Reference for unit 'gpm'	632
17.1	Used units	632
17.2	Overview	632
17.3	Constants, types and variables	632
17.3.1	Constants	632
17.3.2	Types	634
17.3.3	Variables	636
17.4	Procedures and functions	637
17.4.1	Gpm_AnyDouble	637
17.4.2	Gpm_AnySingle	637
17.4.3	Gpm_AnyTriple	637
17.4.4	gpm_close	638
17.4.5	gpm_fitvalues	638
17.4.6	gpm_fitvaluesM	638
17.4.7	gpm_getevent	638
17.4.8	gpm_getsnapshot	640
17.4.9	gpm_lowerroi	640
17.4.10	gpm_open	640
17.4.11	gpm_poproi	641
17.4.12	gpm_pushroi	641
17.4.13	gpm_raiseroi	641

17.4.14	gpm_repeat	641
17.4.15	Gpm_StrictDouble	642
17.4.16	Gpm_StrictSingle	642
17.4.17	Gpm_StrictTriple	642
18	Reference for unit 'Graph'	643
18.1	Overview	643
18.2	Categorized functions: Text and font handling	643
18.3	Categorized functions: Filled drawings	643
18.4	Categorized functions: Drawing primitives	644
18.5	Categorized functions: Color management	644
18.6	Categorized functions: Screen management	645
18.7	Categorized functions: Initialization	645
18.8	Target specific issues: Linux	646
18.9	Target specific issues: DOS	647
18.10	A word about mode selection	647
18.11	Requirements	651
18.12	Constants, types and variables	652
18.12.1	Constants	652
18.12.2	Types	668
18.12.3	Variables	673
18.13	Procedures and functions	676
18.13.1	Arc	676
18.13.2	Bar	676
18.13.3	Bar3D	676
18.13.4	ClearDevice	677
18.13.5	Closegraph	677
18.13.6	DetectGraph	677
18.13.7	DrawPoly	677
18.13.8	Ellipse	678
18.13.9	FillEllipse	678
18.13.10	FillPoly	678
18.13.11	FloodFill	678
18.13.12	GetArcCoords	679
18.13.13	GetAspectRatio	679
18.13.14	GetColor	679
18.13.15	GetDefaultPalette	679
18.13.16	GetDirectVideo	680
18.13.17	GetDriverName	680
18.13.18	GetFillPattern	680

18.13.19	GetFillSettings	680
18.13.20	GetGraphMode	681
18.13.21	GetLineSettings	681
18.13.22	GetMaxColor	681
18.13.23	GetMaxMode	681
18.13.24	GetMaxX	682
18.13.25	GetMaxY	682
18.13.26	GetModeName	682
18.13.27	GetModeRange	682
18.13.28	GetPalette	683
18.13.29	GetPaletteSize	683
18.13.30	GetTextSettings	683
18.13.31	GetViewSettings	683
18.13.32	GetX	684
18.13.33	GetY	684
18.13.34	GraphDefaults	684
18.13.35	GraphErrorMsg	684
18.13.36	GraphResult	685
18.13.37	InitGraph	685
18.13.38	InstallUserDriver	686
18.13.39	InstallUserFont	686
18.13.40	LineRel	686
18.13.41	LineTo	687
18.13.42	MoveRel	687
18.13.43	MoveTo	687
18.13.44	OutText	687
18.13.45	PieSlice	688
18.13.46	queryadapterinfo	688
18.13.47	Rectangle	688
18.13.48	RegisterBGIDriver	688
18.13.49	RegisterBGIfont	689
18.13.50	RestoreCrtMode	689
18.13.51	Sector	689
18.13.52	SetAspectRatio	689
18.13.53	SetColor	690
18.13.54	SetDirectVideo	690
18.13.55	SetFillPattern	690
18.13.56	SetFillStyle	690
18.13.57	SetGraphMode	691
18.13.58	SetLineStyle	691

18.13.59	SetPalette	692
18.13.60	SetTextJustify	692
18.13.61	SetTextStyle	692
18.13.62	SetUserCharSize	693
18.13.63	SetViewPort	693
18.13.64	SetWriteMode	694
18.13.65	TextHeight	694
18.13.66	TextWidth	694
19	Reference for unit 'heaptrc'	695
19.1	Overview	695
19.2	Controlling HeapTrc with environment variables	695
19.3	HeapTrc Usage	696
19.4	Constants, types and variables	697
19.4.1	Constants	697
19.4.2	Types	698
19.5	Procedures and functions	698
19.5.1	CheckPointer	698
19.5.2	DumpHeap	699
19.5.3	SetHeapExtraInfo	699
19.5.4	SetHeapTraceOutput	700
20	Reference for unit 'ipc'	701
20.1	Used units	701
20.2	Overview	701
20.3	Constants, types and variables	701
20.3.1	Constants	701
20.3.2	Types	703
20.4	Procedures and functions	706
20.4.1	ftok	706
20.4.2	msgctl	707
20.4.3	msgget	709
20.4.4	msgrev	710
20.4.5	msgsnd	710
20.4.6	semctl	711
20.4.7	semget	715
20.4.8	semop	716
20.4.9	shmat	717
20.4.10	shmctl	717
20.4.11	shmdt	719
20.4.12	shmget	720

21	Reference for unit 'keyboard'	721
21.1	Overview	721
21.2	Unix specific notes	721
21.3	Writing a keyboard driver	722
21.4	Keyboard scan codes	725
21.5	Constants, types and variables	727
21.5.1	Constants	727
21.5.2	Types	732
21.6	Procedures and functions	733
21.6.1	AddSequence	733
21.6.2	AddSpecialSequence	733
21.6.3	DoneKeyboard	734
21.6.4	FindSequence	734
21.6.5	FunctionKeyName	734
21.6.6	GetKeyboardDriver	735
21.6.7	GetKeyEvent	735
21.6.8	GetKeyEventChar	736
21.6.9	GetKeyEventCode	736
21.6.10	GetKeyEventFlags	737
21.6.11	GetKeyEventShiftState	738
21.6.12	GetKeyEventUniCode	738
21.6.13	InitKeyboard	739
21.6.14	IsFunctionKey	739
21.6.15	KeyEventToString	740
21.6.16	KeyPressed	740
21.6.17	PollKeyEvent	740
21.6.18	PollShiftStateEvent	741
21.6.19	PutKeyEvent	742
21.6.20	RawReadKey	743
21.6.21	RawReadString	743
21.6.22	RestoreStartMode	743
21.6.23	SetKeyboardDriver	743
21.6.24	ShiftStateToString	744
21.6.25	TranslateKeyEvent	744
21.6.26	TranslateKeyEventUniCode	744
22	Reference for unit 'lineinfo'	745
22.1	Overview	745
22.2	Procedures and functions	745
22.2.1	GetLineInfo	745

23 Reference for unit 'Linux'	746
23.1 Used units	746
23.2 Overview	746
23.3 Constants, types and variables	746
23.3.1 Constants	746
23.3.2 Types	760
23.4 Procedures and functions	762
23.4.1 capget	762
23.4.2 capset	762
23.4.3 clock_getres	763
23.4.4 clock_gettime	763
23.4.5 clock_settime	763
23.4.6 clone	763
23.4.7 epoll_create	766
23.4.8 epoll_ctl	766
23.4.9 epoll_wait	767
23.4.10 fdatsync	767
23.4.11 futex	767
23.4.12 futex_op	768
23.4.13 inotify_add_watch	768
23.4.14 inotify_init	769
23.4.15 inotify_init1	769
23.4.16 inotify_rm_watch	770
23.4.17 sched_yield	770
23.4.18 sync_file_range	770
23.4.19 Sysinfo	771
24 Reference for unit 'Infodwrf'	773
24.1 Overview	773
24.2 Procedures and functions	773
24.2.1 GetLineInfo	773
25 Reference for unit 'math'	774
25.1 Used units	774
25.2 Overview	774
25.3 Cash flow functions	774
25.4 Geometrical functions	775
25.5 Statistical functions	775
25.6 Number converting	776
25.7 Exponential and logarithmic functions	776
25.8 Hyperbolic functions	776

25.9	Trigonometric functions	777
25.10	Angle unit conversion	777
25.11	Min/max determination	777
25.12	Constants, types and variables	778
25.12.1	Constants	778
25.12.2	Types	779
25.13	Procedures and functions	780
25.13.1	arccos	780
25.13.2	arccosh	781
25.13.3	arcosh	781
25.13.4	arcsin	782
25.13.5	arsinh	782
25.13.6	arctan2	782
25.13.7	arctanh	783
25.13.8	arsinh	783
25.13.9	artanh	784
25.13.10	ceil	784
25.13.11	ClearExceptions	785
25.13.12	CompareValue	785
25.13.13	cosecant	786
25.13.14	cosh	786
25.13.15	cot	786
25.13.16	cotan	787
25.13.17	csc	787
25.13.18	cycleto rad	787
25.13.19	DegNormalize	788
25.13.20	degtograd	788
25.13.21	degtorad	789
25.13.22	DivMod	789
25.13.23	EnsureRange	789
25.13.24	floor	790
25.13.25	Frexp	790
25.13.26	FutureValue	791
25.13.27	GetExceptionMask	791
25.13.28	GetPrecisionMode	791
25.13.29	GetRoundMode	792
25.13.30	gradtodeg	792
25.13.31	gradtorad	792
25.13.32	hypot	793
25.13.33	ifthen	793

25.13.34 InRange	794
25.13.35 InterestRate	794
25.13.36 intpower	794
25.13.37 IsInfinite	795
25.13.38 IsNan	795
25.13.39 IsZero	795
25.13.40 ldexp	796
25.13.41 lnxp1	796
25.13.42 log10	797
25.13.43 log2	797
25.13.44 logn	798
25.13.45 Max	799
25.13.46 MaxIntValue	799
25.13.47 maxvalue	800
25.13.48 mean	801
25.13.49 meanandstddev	802
25.13.50 Min	803
25.13.51 MinIntValue	803
25.13.52 minvalue	804
25.13.53 momentskewkurtosis	805
25.13.54 norm	806
25.13.55 NumberOfPeriods	807
25.13.56 operator ** (Float, Float): Float	807
25.13.57 operator ** (Int64, Int64): Int64	807
25.13.58 Payment	807
25.13.59 popnstddev	808
25.13.60 popnvariance	808
25.13.61 power	809
25.13.62 PresentValue	810
25.13.63 radto cycle	810
25.13.64 radto deg	811
25.13.65 radto grad	811
25.13.66 randg	812
25.13.67 RandomFrom	812
25.13.68 RandomRange	813
25.13.69 RoundTo	813
25.13.70 SameValue	813
25.13.71 sec	814
25.13.72 secant	814
25.13.73 SetExceptionMask	814

25.13.74	SetPrecisionMode	814
25.13.75	SetRoundMode	815
25.13.76	Sign	815
25.13.77	SimpleRoundTo	815
25.13.78	sincos	815
25.13.79	sinh	816
25.13.80	stddev	817
25.13.81	sum	817
25.13.82	sumInt	818
25.13.83	sumofsquares	818
25.13.84	sumsandsquares	819
25.13.85	tan	820
25.13.86	tanh	821
25.13.87	totalvariance	821
25.13.88	variance	822
25.14	EInvalidArgument	823
25.14.1	Description	823
26	Reference for unit 'matrix'	824
26.1	Overview	824
26.2	Constants, types and variables	825
26.2.1	Types	825
26.3	Procedures and functions	827
26.3.1	operator *(Tmatrix2_double, Double): Tmatrix2_double	827
26.3.2	operator *(Tmatrix2_double, Tmatrix2_double): Tmatrix2_double	827
26.3.3	operator *(Tmatrix2_double, Tvector2_double): Tvector2_double	828
26.3.4	operator *(Tmatrix2_extended, extended): Tmatrix2_extended	828
26.3.5	operator *(Tmatrix2_extended, Tmatrix2_extended): Tmatrix2_extended	828
26.3.6	operator *(Tmatrix2_extended, Tvector2_extended): Tvector2_extended	828
26.3.7	operator *(Tmatrix2_single, single): Tmatrix2_single	829
26.3.8	operator *(Tmatrix2_single, Tmatrix2_single): Tmatrix2_single	829
26.3.9	operator *(Tmatrix2_single, Tvector2_single): Tvector2_single	829
26.3.10	operator *(Tmatrix3_double, Double): Tmatrix3_double	829
26.3.11	operator *(Tmatrix3_double, Tmatrix3_double): Tmatrix3_double	830
26.3.12	operator *(Tmatrix3_double, Tvector3_double): Tvector3_double	830
26.3.13	operator *(Tmatrix3_extended, extended): Tmatrix3_extended	830
26.3.14	operator *(Tmatrix3_extended, Tmatrix3_extended): Tmatrix3_extended	830
26.3.15	operator *(Tmatrix3_extended, Tvector3_extended): Tvector3_extended	831
26.3.16	operator *(Tmatrix3_single, single): Tmatrix3_single	831
26.3.17	operator *(Tmatrix3_single, Tmatrix3_single): Tmatrix3_single	831

26.3.18	operator *(Tmatrix3_single, Tvector3_single): Tvector3_single	832
26.3.19	operator *(Tmatrix4_double, Double): Tmatrix4_double	832
26.3.20	operator *(Tmatrix4_double, Tmatrix4_double): Tmatrix4_double	832
26.3.21	operator *(Tmatrix4_double, Tvector4_double): Tvector4_double	832
26.3.22	operator *(Tmatrix4_extended, extended): Tmatrix4_extended	833
26.3.23	operator *(Tmatrix4_extended, Tmatrix4_extended): Tmatrix4_extended	833
26.3.24	operator *(Tmatrix4_extended, Tvector4_extended): Tvector4_extended	833
26.3.25	operator *(Tmatrix4_single, single): Tmatrix4_single	833
26.3.26	operator *(Tmatrix4_single, Tmatrix4_single): Tmatrix4_single	834
26.3.27	operator *(Tmatrix4_single, Tvector4_single): Tvector4_single	834
26.3.28	operator *(Tvector2_double, Double): Tvector2_double	834
26.3.29	operator *(Tvector2_double, Tvector2_double): Tvector2_double	834
26.3.30	operator *(Tvector2_extended, extended): Tvector2_extended	835
26.3.31	operator *(Tvector2_extended, Tvector2_extended): Tvector2_extended	835
26.3.32	operator *(Tvector2_single, single): Tvector2_single	835
26.3.33	operator *(Tvector2_single, Tvector2_single): Tvector2_single	835
26.3.34	operator *(Tvector3_double, Double): Tvector3_double	836
26.3.35	operator *(Tvector3_double, Tvector3_double): Tvector3_double	836
26.3.36	operator *(Tvector3_extended, extended): Tvector3_extended	836
26.3.37	operator *(Tvector3_extended, Tvector3_extended): Tvector3_extended	836
26.3.38	operator *(Tvector3_single, single): Tvector3_single	837
26.3.39	operator *(Tvector3_single, Tvector3_single): Tvector3_single	837
26.3.40	operator *(Tvector4_double, Double): Tvector4_double	837
26.3.41	operator *(Tvector4_double, Tvector4_double): Tvector4_double	837
26.3.42	operator *(Tvector4_extended, extended): Tvector4_extended	838
26.3.43	operator *(Tvector4_extended, Tvector4_extended): Tvector4_extended	838
26.3.44	operator *(Tvector4_single, single): Tvector4_single	838
26.3.45	operator *(Tvector4_single, Tvector4_single): Tvector4_single	838
26.3.46	operator **(Tvector2_double, Tvector2_double): Double	839
26.3.47	operator **(Tvector2_extended, Tvector2_extended): extended	839
26.3.48	operator **(Tvector2_single, Tvector2_single): single	839
26.3.49	operator **(Tvector3_double, Tvector3_double): Double	839
26.3.50	operator **(Tvector3_extended, Tvector3_extended): extended	840
26.3.51	operator **(Tvector3_single, Tvector3_single): single	840
26.3.52	operator **(Tvector4_double, Tvector4_double): Double	840
26.3.53	operator **(Tvector4_extended, Tvector4_extended): extended	840
26.3.54	operator **(Tvector4_single, Tvector4_single): single	841
26.3.55	operator +(Tmatrix2_double, Double): Tmatrix2_double	841
26.3.56	operator +(Tmatrix2_double, Tmatrix2_double): Tmatrix2_double	841
26.3.57	operator +(Tmatrix2_extended, extended): Tmatrix2_extended	841

26.3.58	operator +(Tmatrix2_extended, Tmatrix2_extended): Tmatrix2_extended	842
26.3.59	operator +(Tmatrix2_single, single): Tmatrix2_single	842
26.3.60	operator +(Tmatrix2_single, Tmatrix2_single): Tmatrix2_single	842
26.3.61	operator +(Tmatrix3_double, Double): Tmatrix3_double	842
26.3.62	operator +(Tmatrix3_double, Tmatrix3_double): Tmatrix3_double	843
26.3.63	operator +(Tmatrix3_extended, extended): Tmatrix3_extended	843
26.3.64	operator +(Tmatrix3_extended, Tmatrix3_extended): Tmatrix3_extended	843
26.3.65	operator +(Tmatrix3_single, single): Tmatrix3_single	843
26.3.66	operator +(Tmatrix3_single, Tmatrix3_single): Tmatrix3_single	844
26.3.67	operator +(Tmatrix4_double, Double): Tmatrix4_double	844
26.3.68	operator +(Tmatrix4_double, Tmatrix4_double): Tmatrix4_double	844
26.3.69	operator +(Tmatrix4_extended, extended): Tmatrix4_extended	844
26.3.70	operator +(Tmatrix4_extended, Tmatrix4_extended): Tmatrix4_extended	845
26.3.71	operator +(Tmatrix4_single, single): Tmatrix4_single	845
26.3.72	operator +(Tmatrix4_single, Tmatrix4_single): Tmatrix4_single	845
26.3.73	operator +(Tvector2_double, Double): Tvector2_double	845
26.3.74	operator +(Tvector2_double, Tvector2_double): Tvector2_double	846
26.3.75	operator +(Tvector2_extended, extended): Tvector2_extended	846
26.3.76	operator +(Tvector2_extended, Tvector2_extended): Tvector2_extended	846
26.3.77	operator +(Tvector2_single, single): Tvector2_single	846
26.3.78	operator +(Tvector2_single, Tvector2_single): Tvector2_single	847
26.3.79	operator +(Tvector3_double, Double): Tvector3_double	847
26.3.80	operator +(Tvector3_double, Tvector3_double): Tvector3_double	847
26.3.81	operator +(Tvector3_extended, extended): Tvector3_extended	847
26.3.82	operator +(Tvector3_extended, Tvector3_extended): Tvector3_extended	848
26.3.83	operator +(Tvector3_single, single): Tvector3_single	848
26.3.84	operator +(Tvector3_single, Tvector3_single): Tvector3_single	848
26.3.85	operator +(Tvector4_double, Double): Tvector4_double	848
26.3.86	operator +(Tvector4_double, Tvector4_double): Tvector4_double	849
26.3.87	operator +(Tvector4_extended, extended): Tvector4_extended	849
26.3.88	operator +(Tvector4_extended, Tvector4_extended): Tvector4_extended	849
26.3.89	operator +(Tvector4_single, single): Tvector4_single	849
26.3.90	operator +(Tvector4_single, Tvector4_single): Tvector4_single	850
26.3.91	operator -(Tmatrix2_double): Tmatrix2_double	850
26.3.92	operator -(Tmatrix2_double, Double): Tmatrix2_double	850
26.3.93	operator -(Tmatrix2_double, Tmatrix2_double): Tmatrix2_double	850
26.3.94	operator -(Tmatrix2_extended): Tmatrix2_extended	851
26.3.95	operator -(Tmatrix2_extended, extended): Tmatrix2_extended	851
26.3.96	operator -(Tmatrix2_extended, Tmatrix2_extended): Tmatrix2_extended	851
26.3.97	operator -(Tmatrix2_single): Tmatrix2_single	851

26.3.98	operator -(Tmatrix2_single, single): Tmatrix2_single	852
26.3.99	operator -(Tmatrix2_single, Tmatrix2_single): Tmatrix2_single	852
26.3.100	operator -(Tmatrix3_double): Tmatrix3_double	852
26.3.101	operator -(Tmatrix3_double, Double): Tmatrix3_double	852
26.3.102	operator -(Tmatrix3_double, Tmatrix3_double): Tmatrix3_double	853
26.3.103	operator -(Tmatrix3_extended): Tmatrix3_extended	853
26.3.104	operator -(Tmatrix3_extended, extended): Tmatrix3_extended	853
26.3.105	operator -(Tmatrix3_extended, Tmatrix3_extended): Tmatrix3_extended	853
26.3.106	operator -(Tmatrix3_single): Tmatrix3_single	854
26.3.107	operator -(Tmatrix3_single, single): Tmatrix3_single	854
26.3.108	operator -(Tmatrix3_single, Tmatrix3_single): Tmatrix3_single	854
26.3.109	operator -(Tmatrix4_double): Tmatrix4_double	854
26.3.110	operator -(Tmatrix4_double, Double): Tmatrix4_double	855
26.3.111	operator -(Tmatrix4_double, Tmatrix4_double): Tmatrix4_double	855
26.3.112	operator -(Tmatrix4_extended): Tmatrix4_extended	855
26.3.113	operator -(Tmatrix4_extended, extended): Tmatrix4_extended	855
26.3.114	operator -(Tmatrix4_extended, Tmatrix4_extended): Tmatrix4_extended	856
26.3.115	operator -(Tmatrix4_single): Tmatrix4_single	856
26.3.116	operator -(Tmatrix4_single, single): Tmatrix4_single	856
26.3.117	operator -(Tmatrix4_single, Tmatrix4_single): Tmatrix4_single	856
26.3.118	operator -(Tvector2_double): Tvector2_double	857
26.3.119	operator -(Tvector2_double, Double): Tvector2_double	857
26.3.120	operator -(Tvector2_double, Tvector2_double): Tvector2_double	857
26.3.121	operator -(Tvector2_extended): Tvector2_extended	857
26.3.122	operator -(Tvector2_extended, extended): Tvector2_extended	858
26.3.123	operator -(Tvector2_extended, Tvector2_extended): Tvector2_extended	858
26.3.124	operator -(Tvector2_single): Tvector2_single	858
26.3.125	operator -(Tvector2_single, single): Tvector2_single	858
26.3.126	operator -(Tvector2_single, Tvector2_single): Tvector2_single	859
26.3.127	operator -(Tvector3_double): Tvector3_double	859
26.3.128	operator -(Tvector3_double, Double): Tvector3_double	859
26.3.129	operator -(Tvector3_double, Tvector3_double): Tvector3_double	859
26.3.130	operator -(Tvector3_extended): Tvector3_extended	860
26.3.131	operator -(Tvector3_extended, extended): Tvector3_extended	860
26.3.132	operator -(Tvector3_extended, Tvector3_extended): Tvector3_extended	860
26.3.133	operator -(Tvector3_single): Tvector3_single	860
26.3.134	operator -(Tvector3_single, single): Tvector3_single	861
26.3.135	operator -(Tvector3_single, Tvector3_single): Tvector3_single	861
26.3.136	operator -(Tvector4_double): Tvector4_double	861
26.3.137	operator -(Tvector4_double, Double): Tvector4_double	861

26.3.138	operator -(Tvector4_double, Tvector4_double): Tvector4_double	862
26.3.139	operator -(Tvector4_extended): Tvector4_extended	862
26.3.140	operator -(Tvector4_extended, extended): Tvector4_extended	862
26.3.141	operator -(Tvector4_extended, Tvector4_extended): Tvector4_extended	862
26.3.142	operator -(Tvector4_single): Tvector4_single	863
26.3.143	operator -(Tvector4_single, single): Tvector4_single	863
26.3.144	operator -(Tvector4_single, Tvector4_single): Tvector4_single	863
26.3.145	operator /(Tmatrix2_double, Double): Tmatrix2_double	863
26.3.146	operator /(Tmatrix2_extended, extended): Tmatrix2_extended	864
26.3.147	operator /(Tmatrix2_single, single): Tmatrix2_single	864
26.3.148	operator /(Tmatrix3_double, Double): Tmatrix3_double	864
26.3.149	operator /(Tmatrix3_extended, extended): Tmatrix3_extended	864
26.3.150	operator /(Tmatrix3_single, single): Tmatrix3_single	865
26.3.151	operator /(Tmatrix4_double, Double): Tmatrix4_double	865
26.3.152	operator /(Tmatrix4_extended, extended): Tmatrix4_extended	865
26.3.153	operator /(Tmatrix4_single, single): Tmatrix4_single	865
26.3.154	operator /(Tvector2_double, Double): Tvector2_double	866
26.3.155	operator /(Tvector2_extended, extended): Tvector2_extended	866
26.3.156	operator /(Tvector2_single, single): Tvector2_single	866
26.3.157	operator /(Tvector3_double, Double): Tvector3_double	866
26.3.158	operator /(Tvector3_extended, extended): Tvector3_extended	867
26.3.159	operator /(Tvector3_single, single): Tvector3_single	867
26.3.160	operator /(Tvector4_double, Double): Tvector4_double	867
26.3.161	operator /(Tvector4_extended, extended): Tvector4_extended	867
26.3.162	operator /(Tvector4_single, single): Tvector4_single	868
26.3.163	operator :=(Tmatrix2_double): Tmatrix2_extended	868
26.3.164	operator :=(Tmatrix2_double): Tmatrix2_single	868
26.3.165	operator :=(Tmatrix2_double): Tmatrix3_double	868
26.3.166	operator :=(Tmatrix2_double): Tmatrix3_extended	869
26.3.167	operator :=(Tmatrix2_double): Tmatrix3_single	869
26.3.168	operator :=(Tmatrix2_double): Tmatrix4_double	869
26.3.169	operator :=(Tmatrix2_double): Tmatrix4_extended	869
26.3.170	operator :=(Tmatrix2_double): Tmatrix4_single	870
26.3.171	operator :=(Tmatrix2_extended): Tmatrix2_double	870
26.3.172	operator :=(Tmatrix2_extended): Tmatrix2_single	870
26.3.173	operator :=(Tmatrix2_extended): Tmatrix3_double	870
26.3.174	operator :=(Tmatrix2_extended): Tmatrix3_extended	871
26.3.175	operator :=(Tmatrix2_extended): Tmatrix3_single	871
26.3.176	operator :=(Tmatrix2_extended): Tmatrix4_double	871
26.3.177	operator :=(Tmatrix2_extended): Tmatrix4_extended	871

26.3.178	operator :=(Tmatrix2_extended): Tmatrix4_single	872
26.3.179	operator :=(Tmatrix2_single): Tmatrix2_double	872
26.3.180	operator :=(Tmatrix2_single): Tmatrix2_extended	872
26.3.181	operator :=(Tmatrix2_single): Tmatrix3_double	872
26.3.182	operator :=(Tmatrix2_single): Tmatrix3_extended	873
26.3.183	operator :=(Tmatrix2_single): Tmatrix3_single	873
26.3.184	operator :=(Tmatrix2_single): Tmatrix4_double	873
26.3.185	operator :=(Tmatrix2_single): Tmatrix4_extended	873
26.3.186	operator :=(Tmatrix2_single): Tmatrix4_single	874
26.3.187	operator :=(Tmatrix3_double): Tmatrix2_double	874
26.3.188	operator :=(Tmatrix3_double): Tmatrix2_extended	874
26.3.189	operator :=(Tmatrix3_double): Tmatrix2_single	874
26.3.190	operator :=(Tmatrix3_double): Tmatrix3_extended	875
26.3.191	operator :=(Tmatrix3_double): Tmatrix3_single	875
26.3.192	operator :=(Tmatrix3_double): Tmatrix4_double	875
26.3.193	operator :=(Tmatrix3_double): Tmatrix4_extended	875
26.3.194	operator :=(Tmatrix3_double): Tmatrix4_single	876
26.3.195	operator :=(Tmatrix3_extended): Tmatrix2_double	876
26.3.196	operator :=(Tmatrix3_extended): Tmatrix2_extended	876
26.3.197	operator :=(Tmatrix3_extended): Tmatrix2_single	877
26.3.198	operator :=(Tmatrix3_extended): Tmatrix3_double	877
26.3.199	operator :=(Tmatrix3_extended): Tmatrix3_single	877
26.3.200	operator :=(Tmatrix3_extended): Tmatrix4_double	877
26.3.201	operator :=(Tmatrix3_extended): Tmatrix4_extended	878
26.3.202	operator :=(Tmatrix3_extended): Tmatrix4_single	878
26.3.203	operator :=(Tmatrix3_single): Tmatrix2_double	878
26.3.204	operator :=(Tmatrix3_single): Tmatrix2_extended	878
26.3.205	operator :=(Tmatrix3_single): Tmatrix2_single	879
26.3.206	operator :=(Tmatrix3_single): Tmatrix3_double	879
26.3.207	operator :=(Tmatrix3_single): Tmatrix3_extended	879
26.3.208	operator :=(Tmatrix3_single): Tmatrix4_double	879
26.3.209	operator :=(Tmatrix3_single): Tmatrix4_extended	880
26.3.210	operator :=(Tmatrix3_single): Tmatrix4_single	880
26.3.211	operator :=(Tmatrix4_double): Tmatrix2_double	880
26.3.212	operator :=(Tmatrix4_double): Tmatrix2_extended	880
26.3.213	operator :=(Tmatrix4_double): Tmatrix2_single	881
26.3.214	operator :=(Tmatrix4_double): Tmatrix3_double	881
26.3.215	operator :=(Tmatrix4_double): Tmatrix3_extended	881
26.3.216	operator :=(Tmatrix4_double): Tmatrix3_single	881
26.3.217	operator :=(Tmatrix4_double): Tmatrix4_extended	882

26.3.218	operator :=(Tmatrix4_double): Tmatrix4_single	882
26.3.219	operator :=(Tmatrix4_extended): Tmatrix2_double	882
26.3.220	operator :=(Tmatrix4_extended): Tmatrix2_extended	882
26.3.221	operator :=(Tmatrix4_extended): Tmatrix2_single	883
26.3.222	operator :=(Tmatrix4_extended): Tmatrix3_double	883
26.3.223	operator :=(Tmatrix4_extended): Tmatrix3_extended	883
26.3.224	operator :=(Tmatrix4_extended): Tmatrix3_single	883
26.3.225	operator :=(Tmatrix4_extended): Tmatrix4_double	884
26.3.226	operator :=(Tmatrix4_extended): Tmatrix4_single	884
26.3.227	operator :=(Tmatrix4_single): Tmatrix2_double	884
26.3.228	operator :=(Tmatrix4_single): Tmatrix2_extended	884
26.3.229	operator :=(Tmatrix4_single): Tmatrix2_single	885
26.3.230	operator :=(Tmatrix4_single): Tmatrix3_double	885
26.3.231	operator :=(Tmatrix4_single): Tmatrix3_extended	885
26.3.232	operator :=(Tmatrix4_single): Tmatrix3_single	885
26.3.233	operator :=(Tmatrix4_single): Tmatrix4_double	886
26.3.234	operator :=(Tmatrix4_single): Tmatrix4_extended	886
26.3.235	operator :=(Tvector2_double): Tvector2_extended	886
26.3.236	operator :=(Tvector2_double): Tvector2_single	886
26.3.237	operator :=(Tvector2_double): Tvector3_double	887
26.3.238	operator :=(Tvector2_double): Tvector3_extended	887
26.3.239	operator :=(Tvector2_double): Tvector3_single	887
26.3.240	operator :=(Tvector2_double): Tvector4_double	887
26.3.241	operator :=(Tvector2_double): Tvector4_extended	888
26.3.242	operator :=(Tvector2_double): Tvector4_single	888
26.3.243	operator :=(Tvector2_extended): Tvector2_double	888
26.3.244	operator :=(Tvector2_extended): Tvector2_single	888
26.3.245	operator :=(Tvector2_extended): Tvector3_double	889
26.3.246	operator :=(Tvector2_extended): Tvector3_extended	889
26.3.247	operator :=(Tvector2_extended): Tvector3_single	889
26.3.248	operator :=(Tvector2_extended): Tvector4_double	889
26.3.249	operator :=(Tvector2_extended): Tvector4_extended	890
26.3.250	operator :=(Tvector2_extended): Tvector4_single	890
26.3.251	operator :=(Tvector2_single): Tvector2_double	890
26.3.252	operator :=(Tvector2_single): Tvector2_extended	890
26.3.253	operator :=(Tvector2_single): Tvector3_double	891
26.3.254	operator :=(Tvector2_single): Tvector3_extended	891
26.3.255	operator :=(Tvector2_single): Tvector3_single	891
26.3.256	operator :=(Tvector2_single): Tvector4_double	891
26.3.257	operator :=(Tvector2_single): Tvector4_extended	892

26.3.258	operator :=(Tvector2_single): Tvector4_single	892
26.3.259	operator :=(Tvector3_double): Tvector2_double	892
26.3.260	operator :=(Tvector3_double): Tvector2_extended	892
26.3.261	operator :=(Tvector3_double): Tvector2_single	893
26.3.262	operator :=(Tvector3_double): Tvector3_extended	893
26.3.263	operator :=(Tvector3_double): Tvector3_single	893
26.3.264	operator :=(Tvector3_double): Tvector4_double	893
26.3.265	operator :=(Tvector3_double): Tvector4_extended	894
26.3.266	operator :=(Tvector3_double): Tvector4_single	894
26.3.267	operator :=(Tvector3_extended): Tvector2_double	894
26.3.268	operator :=(Tvector3_extended): Tvector2_extended	894
26.3.269	operator :=(Tvector3_extended): Tvector2_single	895
26.3.270	operator :=(Tvector3_extended): Tvector3_double	895
26.3.271	operator :=(Tvector3_extended): Tvector3_single	895
26.3.272	operator :=(Tvector3_extended): Tvector4_double	895
26.3.273	operator :=(Tvector3_extended): Tvector4_extended	896
26.3.274	operator :=(Tvector3_extended): Tvector4_single	896
26.3.275	operator :=(Tvector3_single): Tvector2_double	896
26.3.276	operator :=(Tvector3_single): Tvector2_extended	896
26.3.277	operator :=(Tvector3_single): Tvector2_single	897
26.3.278	operator :=(Tvector3_single): Tvector3_double	897
26.3.279	operator :=(Tvector3_single): Tvector3_extended	897
26.3.280	operator :=(Tvector3_single): Tvector4_double	897
26.3.281	operator :=(Tvector3_single): Tvector4_extended	898
26.3.282	operator :=(Tvector3_single): Tvector4_single	898
26.3.283	operator :=(Tvector4_double): Tvector2_double	898
26.3.284	operator :=(Tvector4_double): Tvector2_extended	898
26.3.285	operator :=(Tvector4_double): Tvector2_single	899
26.3.286	operator :=(Tvector4_double): Tvector3_double	899
26.3.287	operator :=(Tvector4_double): Tvector3_extended	899
26.3.288	operator :=(Tvector4_double): Tvector3_single	899
26.3.289	operator :=(Tvector4_double): Tvector4_extended	900
26.3.290	operator :=(Tvector4_double): Tvector4_single	900
26.3.291	operator :=(Tvector4_extended): Tvector2_double	900
26.3.292	operator :=(Tvector4_extended): Tvector2_extended	901
26.3.293	operator :=(Tvector4_extended): Tvector2_single	901
26.3.294	operator :=(Tvector4_extended): Tvector3_double	901
26.3.295	operator :=(Tvector4_extended): Tvector3_extended	901
26.3.296	operator :=(Tvector4_extended): Tvector3_single	902
26.3.297	operator :=(Tvector4_extended): Tvector4_double	902

26.3.298	operator :=(Tvector4_extended): Tvector4_single	902
26.3.299	operator :=(Tvector4_single): Tvector2_double	902
26.3.300	operator :=(Tvector4_single): Tvector2_extended	903
26.3.301	operator :=(Tvector4_single): Tvector2_single	903
26.3.302	operator :=(Tvector4_single): Tvector3_double	903
26.3.303	operator :=(Tvector4_single): Tvector3_extended	904
26.3.304	operator :=(Tvector4_single): Tvector3_single	904
26.3.305	operator :=(Tvector4_single): Tvector4_double	904
26.3.306	operator :=(Tvector4_single): Tvector4_extended	904
26.3.307	operator ><(Tvector3_double, Tvector3_double): Tvector3_double	905
26.3.308	operator ><(Tvector3_extended, Tvector3_extended): Tvector3_extended	905
26.3.309	operator ><(Tvector3_single, Tvector3_single): Tvector3_single	905
26.4	Tmatrix2_double	906
26.4.1	Description	906
26.4.2	Method overview	906
26.4.3	Tmatrix2_double.init_zero	906
26.4.4	Tmatrix2_double.init_identity	906
26.4.5	Tmatrix2_double.init	906
26.4.6	Tmatrix2_double.get_column	907
26.4.7	Tmatrix2_double.get_row	907
26.4.8	Tmatrix2_double.set_column	907
26.4.9	Tmatrix2_double.set_row	907
26.4.10	Tmatrix2_double.determinant	907
26.4.11	Tmatrix2_double.inverse	907
26.4.12	Tmatrix2_double.transpose	908
26.5	Tmatrix2_extended	908
26.5.1	Description	908
26.5.2	Method overview	908
26.5.3	Tmatrix2_extended.init_zero	908
26.5.4	Tmatrix2_extended.init_identity	908
26.5.5	Tmatrix2_extended.init	909
26.5.6	Tmatrix2_extended.get_column	909
26.5.7	Tmatrix2_extended.get_row	909
26.5.8	Tmatrix2_extended.set_column	909
26.5.9	Tmatrix2_extended.set_row	909
26.5.10	Tmatrix2_extended.determinant	909
26.5.11	Tmatrix2_extended.inverse	910
26.5.12	Tmatrix2_extended.transpose	910
26.6	Tmatrix2_single	910
26.6.1	Description	910

26.6.2	Method overview	910
26.6.3	Tmatrix2_single.init_zero	910
26.6.4	Tmatrix2_single.init_identity	911
26.6.5	Tmatrix2_single.init	911
26.6.6	Tmatrix2_single.get_column	911
26.6.7	Tmatrix2_single.get_row	911
26.6.8	Tmatrix2_single.set_column	911
26.6.9	Tmatrix2_single.set_row	912
26.6.10	Tmatrix2_single.determinant	912
26.6.11	Tmatrix2_single.inverse	912
26.6.12	Tmatrix2_single.transpose	912
26.7	Tmatrix3_double	912
26.7.1	Description	912
26.7.2	Method overview	913
26.7.3	Tmatrix3_double.init_zero	913
26.7.4	Tmatrix3_double.init_identity	913
26.7.5	Tmatrix3_double.init	913
26.7.6	Tmatrix3_double.get_column	913
26.7.7	Tmatrix3_double.get_row	914
26.7.8	Tmatrix3_double.set_column	914
26.7.9	Tmatrix3_double.set_row	914
26.7.10	Tmatrix3_double.determinant	914
26.7.11	Tmatrix3_double.inverse	914
26.7.12	Tmatrix3_double.transpose	914
26.8	Tmatrix3_extended	915
26.8.1	Description	915
26.8.2	Method overview	915
26.8.3	Tmatrix3_extended.init_zero	915
26.8.4	Tmatrix3_extended.init_identity	915
26.8.5	Tmatrix3_extended.init	915
26.8.6	Tmatrix3_extended.get_column	916
26.8.7	Tmatrix3_extended.get_row	916
26.8.8	Tmatrix3_extended.set_column	916
26.8.9	Tmatrix3_extended.set_row	916
26.8.10	Tmatrix3_extended.determinant	916
26.8.11	Tmatrix3_extended.inverse	916
26.8.12	Tmatrix3_extended.transpose	917
26.9	Tmatrix3_single	917
26.9.1	Description	917
26.9.2	Method overview	917

26.9.3	Tmatrix3_single.init_zero	917
26.9.4	Tmatrix3_single.init_identity	917
26.9.5	Tmatrix3_single.init	918
26.9.6	Tmatrix3_single.get_column	918
26.9.7	Tmatrix3_single.get_row	918
26.9.8	Tmatrix3_single.set_column	918
26.9.9	Tmatrix3_single.set_row	918
26.9.10	Tmatrix3_single.determinant	919
26.9.11	Tmatrix3_single.inverse	919
26.9.12	Tmatrix3_single.transpose	919
26.10	Tmatrix4_double	919
26.10.1	Description	919
26.10.2	Method overview	919
26.10.3	Tmatrix4_double.init_zero	920
26.10.4	Tmatrix4_double.init_identity	920
26.10.5	Tmatrix4_double.init	920
26.10.6	Tmatrix4_double.get_column	920
26.10.7	Tmatrix4_double.get_row	920
26.10.8	Tmatrix4_double.set_column	921
26.10.9	Tmatrix4_double.set_row	921
26.10.10	Tmatrix4_double.determinant	921
26.10.11	Tmatrix4_double.inverse	921
26.10.12	Tmatrix4_double.transpose	921
26.11	Tmatrix4_extended	922
26.11.1	Description	922
26.11.2	Method overview	922
26.11.3	Tmatrix4_extended.init_zero	922
26.11.4	Tmatrix4_extended.init_identity	922
26.11.5	Tmatrix4_extended.init	922
26.11.6	Tmatrix4_extended.get_column	923
26.11.7	Tmatrix4_extended.get_row	923
26.11.8	Tmatrix4_extended.set_column	923
26.11.9	Tmatrix4_extended.set_row	923
26.11.10	Tmatrix4_extended.determinant	923
26.11.11	Tmatrix4_extended.inverse	924
26.11.12	Tmatrix4_extended.transpose	924
26.12	Tmatrix4_single	924
26.12.1	Description	924
26.12.2	Method overview	924
26.12.3	Tmatrix4_single.init_zero	924

26.12.4	Tmatrix4_single.init_identity	925
26.12.5	Tmatrix4_single.init	925
26.12.6	Tmatrix4_single.get_column	925
26.12.7	Tmatrix4_single.get_row	925
26.12.8	Tmatrix4_single.set_column	925
26.12.9	Tmatrix4_single.set_row	926
26.12.10	Tmatrix4_single.determinant	926
26.12.11	Tmatrix4_single.inverse	926
26.12.12	Tmatrix4_single.transpose	926
26.13	Tvector2_double	926
26.13.1	Description	926
26.13.2	Method overview	927
26.13.3	Tvector2_double.init_zero	927
26.13.4	Tvector2_double.init_one	927
26.13.5	Tvector2_double.init	927
26.13.6	Tvector2_double.length	927
26.13.7	Tvector2_double.squared_length	927
26.14	Tvector2_extended	928
26.14.1	Description	928
26.14.2	Method overview	928
26.14.3	Tvector2_extended.init_zero	928
26.14.4	Tvector2_extended.init_one	928
26.14.5	Tvector2_extended.init	928
26.14.6	Tvector2_extended.length	928
26.14.7	Tvector2_extended.squared_length	929
26.15	Tvector2_single	929
26.15.1	Description	929
26.15.2	Method overview	929
26.15.3	Tvector2_single.init_zero	929
26.15.4	Tvector2_single.init_one	929
26.15.5	Tvector2_single.init	929
26.15.6	Tvector2_single.length	930
26.15.7	Tvector2_single.squared_length	930
26.16	Tvector3_double	930
26.16.1	Description	930
26.16.2	Method overview	930
26.16.3	Tvector3_double.init_zero	930
26.16.4	Tvector3_double.init_one	930
26.16.5	Tvector3_double.init	931
26.16.6	Tvector3_double.length	931

26.16.7	Tvector3_double.squared_length	931
26.17	Tvector3_extended	931
26.17.1	Description	931
26.17.2	Method overview	931
26.17.3	Tvector3_extended.init_zero	931
26.17.4	Tvector3_extended.init_one	932
26.17.5	Tvector3_extended.init	932
26.17.6	Tvector3_extended.length	932
26.17.7	Tvector3_extended.squared_length	932
26.18	Tvector3_single	932
26.18.1	Description	932
26.18.2	Method overview	932
26.18.3	Tvector3_single.init_zero	933
26.18.4	Tvector3_single.init_one	933
26.18.5	Tvector3_single.init	933
26.18.6	Tvector3_single.length	933
26.18.7	Tvector3_single.squared_length	933
26.19	Tvector4_double	933
26.19.1	Description	933
26.19.2	Method overview	934
26.19.3	Tvector4_double.init_zero	934
26.19.4	Tvector4_double.init_one	934
26.19.5	Tvector4_double.init	934
26.19.6	Tvector4_double.length	934
26.19.7	Tvector4_double.squared_length	934
26.20	Tvector4_extended	935
26.20.1	Description	935
26.20.2	Method overview	935
26.20.3	Tvector4_extended.init_zero	935
26.20.4	Tvector4_extended.init_one	935
26.20.5	Tvector4_extended.init	935
26.20.6	Tvector4_extended.length	935
26.20.7	Tvector4_extended.squared_length	936
26.21	Tvector4_single	936
26.21.1	Description	936
26.21.2	Method overview	936
26.21.3	Tvector4_single.init_zero	936
26.21.4	Tvector4_single.init_one	936
26.21.5	Tvector4_single.init	936
26.21.6	Tvector4_single.length	937

26.21.7	Tvector4_single.squared_length	937
27	Reference for unit 'mmx'	938
27.1	Overview	938
27.2	Constants, types and variables	938
27.2.1	Constants	938
27.2.2	Types	939
27.3	Procedures and functions	940
27.3.1	emms	940
27.3.2	femms	940
28	Reference for unit 'Mouse'	941
28.1	Overview	941
28.2	Writing a custom mouse driver	941
28.3	Constants, types and variables	943
28.3.1	Constants	943
28.3.2	Types	944
28.3.3	Variables	945
28.4	Procedures and functions	945
28.4.1	DetectMouse	945
28.4.2	DoneMouse	946
28.4.3	GetMouseButtons	946
28.4.4	GetMouseDriver	947
28.4.5	GetMouseEvent	947
28.4.6	GetMouseX	948
28.4.7	GetMouseY	948
28.4.8	HideMouse	949
28.4.9	InitMouse	949
28.4.10	PollMouseEvent	950
28.4.11	PutMouseEvent	950
28.4.12	SetMouseDriver	950
28.4.13	SetMouseXY	951
28.4.14	ShowMouse	951
29	Reference for unit 'Objects'	952
29.1	Overview	952
29.2	Constants, types and variables	952
29.2.1	Constants	952
29.2.2	Types	954
29.2.3	Variables	958
29.3	Procedures and functions	958

29.3.1	Abstract	958
29.3.2	CallPointerConstructor	958
29.3.3	CallPointerLocal	959
29.3.4	CallPointerMethod	959
29.3.5	CallPointerMethodLocal	959
29.3.6	CallVoidConstructor	960
29.3.7	CallVoidLocal	960
29.3.8	CallVoidMethod	960
29.3.9	CallVoidMethodLocal	961
29.3.10	DisposeStr	961
29.3.11	LongDiv	961
29.3.12	LongMul	961
29.3.13	NewStr	962
29.3.14	RegisterObjects	962
29.3.15	RegisterType	963
29.3.16	SetStr	964
29.4	TBufStream	965
29.4.1	Description	965
29.4.2	Method overview	965
29.4.3	TBufStream.Init	965
29.4.4	TBufStream.Done	966
29.4.5	TBufStream.Close	966
29.4.6	TBufStream.Flush	966
29.4.7	TBufStream.Truncate	967
29.4.8	TBufStream.Seek	967
29.4.9	TBufStream.Open	968
29.4.10	TBufStream.Read	968
29.4.11	TBufStream.Write	968
29.5	TCollection	969
29.5.1	Description	969
29.5.2	Method overview	969
29.5.3	TCollection.Init	969
29.5.4	TCollection.Load	970
29.5.5	TCollection.Done	970
29.5.6	TCollection.At	971
29.5.7	TCollection.IndexOf	971
29.5.8	TCollection.GetItem	972
29.5.9	TCollection.LastThat	973
29.5.10	TCollection.FirstThat	973
29.5.11	TCollection.Pack	974

29.5.12	TCollection.FreeAll	975
29.5.13	TCollection.DeleteAll	976
29.5.14	TCollection.Free	977
29.5.15	TCollection.Insert	977
29.5.16	TCollection.Delete	978
29.5.17	TCollection.AtFree	978
29.5.18	TCollection.FreeItem	979
29.5.19	TCollection.AtDelete	979
29.5.20	TCollection.ForEach	980
29.5.21	TCollection.SetLimit	981
29.5.22	TCollection.Error	981
29.5.23	TCollection.AtPut	982
29.5.24	TCollection.AtInsert	982
29.5.25	TCollection.Store	983
29.5.26	TCollection.PutItem	983
29.6	TDosStream	983
29.6.1	Description	983
29.6.2	Method overview	984
29.6.3	TDosStream.Init	984
29.6.4	TDosStream.Done	984
29.6.5	TDosStream.Close	985
29.6.6	TDosStream.Truncate	985
29.6.7	TDosStream.Seek	986
29.6.8	TDosStream.Open	987
29.6.9	TDosStream.Read	987
29.6.10	TDosStream.Write	988
29.7	TMemoryStream	988
29.7.1	Description	988
29.7.2	Method overview	988
29.7.3	TMemoryStream.Init	988
29.7.4	TMemoryStream.Done	989
29.7.5	TMemoryStream.Truncate	989
29.7.6	TMemoryStream.Read	990
29.7.7	TMemoryStream.Write	990
29.8	TObject	990
29.8.1	Description	990
29.8.2	Method overview	990
29.8.3	TObject.Init	990
29.8.4	TObject.Free	991
29.8.5	TObject.Is_Object	991

29.8.6	TObject.Done	992
29.9	TPoint	992
29.9.1	Description	992
29.10	TRect	992
29.10.1	Description	992
29.10.2	Method overview	992
29.10.3	TRect.Empty	993
29.10.4	TRect.Equals	994
29.10.5	TRect.Contains	994
29.10.6	TRect.Copy	994
29.10.7	TRect.Union	995
29.10.8	TRect.Intersect	995
29.10.9	TRect.Move	996
29.10.10	TRect.Grow	997
29.10.11	TRect.Assign	997
29.11	TResourceCollection	998
29.11.1	Description	998
29.11.2	Method overview	998
29.11.3	TResourceCollection.KeyOf	998
29.11.4	TResourceCollection.GetItem	999
29.11.5	TResourceCollection.FreeItem	999
29.11.6	TResourceCollection.PutItem	999
29.12	TResourceFile	999
29.12.1	Description	999
29.12.2	Method overview	1000
29.12.3	TResourceFile.Init	1000
29.12.4	TResourceFile.Done	1000
29.12.5	TResourceFile.Count	1000
29.12.6	TResourceFile.KeyAt	1001
29.12.7	TResourceFile.Get	1001
29.12.8	TResourceFile.SwitchTo	1001
29.12.9	TResourceFile.Flush	1001
29.12.10	TResourceFile.Delete	1002
29.12.11	TResourceFile.Put	1002
29.13	TSortedCollection	1002
29.13.1	Description	1002
29.13.2	Method overview	1003
29.13.3	TSortedCollection.Init	1003
29.13.4	TSortedCollection.Load	1003
29.13.5	TSortedCollection.KeyOf	1003

29.13.6	TSortedCollection.IndexOf	1004
29.13.7	TSortedCollection.Compare	1004
29.13.8	TSortedCollection.Search	1005
29.13.9	TSortedCollection.Insert	1006
29.13.10	TSortedCollection.Store	1007
29.14	TStrCollection	1008
29.14.1	Description	1008
29.14.2	Method overview	1008
29.14.3	TStrCollection.Compare	1008
29.14.4	TStrCollection.GetItem	1009
29.14.5	TStrCollection.FreeItem	1009
29.14.6	TStrCollection.PutItem	1009
29.15	TStream	1010
29.15.1	Description	1010
29.15.2	Method overview	1010
29.15.3	TStream.Init	1010
29.15.4	TStream.Get	1010
29.15.5	TStream.StrRead	1011
29.15.6	TStream.GetPos	1012
29.15.7	TStream.GetSize	1012
29.15.8	TStream.ReadStr	1013
29.15.9	TStream.Open	1014
29.15.10	TStream.Close	1014
29.15.11	TStream.Reset	1014
29.15.12	TStream.Flush	1015
29.15.13	TStream.Truncate	1015
29.15.14	TStream.Put	1015
29.15.15	TStream.StrWrite	1016
29.15.16	TStream.WriteStr	1016
29.15.17	TStream.Seek	1016
29.15.18	TStream.Error	1016
29.15.19	TStream.Read	1017
29.15.20	TStream.Write	1017
29.15.21	TStream.CopyFrom	1018
29.16	TStringCollection	1018
29.16.1	Description	1018
29.16.2	Method overview	1019
29.16.3	TStringCollection.GetItem	1019
29.16.4	TStringCollection.Compare	1019
29.16.5	TStringCollection.FreeItem	1020

29.16.6	TStringCollection.PutItem	1020
29.17	TStringList	1020
29.17.1	Description	1020
29.17.2	Method overview	1021
29.17.3	TStringList.Load	1021
29.17.4	TStringList.Done	1021
29.17.5	TStringList.Get	1021
29.18	TStrListMaker	1022
29.18.1	Description	1022
29.18.2	Method overview	1022
29.18.3	TStrListMaker.Init	1022
29.18.4	TStrListMaker.Done	1022
29.18.5	TStrListMaker.Put	1022
29.18.6	TStrListMaker.Store	1023
29.19	TUnSortedStrCollection	1023
29.19.1	Description	1023
29.19.2	Method overview	1023
29.19.3	TUnSortedStrCollection.Insert	1023
30	Reference for unit 'objpas'	1025
30.1	Overview	1025
30.2	Constants, types and variables	1025
30.2.1	Constants	1025
30.2.2	Types	1025
31	Reference for unit 'ports'	1027
31.1	Overview	1027
31.2	Constants, types and variables	1027
31.2.1	Variables	1027
31.3	tport	1028
31.3.1	Description	1028
31.3.2	Property overview	1028
31.3.3	tport.pp	1028
31.4	tportl	1029
31.4.1	Description	1029
31.4.2	Property overview	1029
31.4.3	tportl.pp	1029
31.5	tportw	1029
31.5.1	Description	1029
31.5.2	Property overview	1029
31.5.3	tportw.pp	1029

32 Reference for unit 'printer'	1030
32.1 Overview	1030
32.2 Constants, types and variables	1030
32.2.1 Variables	1030
32.3 Procedures and functions	1030
32.3.1 AssignLst	1030
32.3.2 InitPrinter	1031
32.3.3 IsLstAvailable	1031
 33 Reference for unit 'Sockets'	 1032
33.1 Used units	1032
33.2 Overview	1032
33.3 Constants, types and variables	1032
33.3.1 Constants	1032
33.3.2 Types	1053
33.4 Procedures and functions	1056
33.4.1 Accept	1056
33.4.2 Bind	1058
33.4.3 CloseSocket	1058
33.4.4 Connect	1058
33.4.5 faccept	1060
33.4.6 fbind	1061
33.4.7 fconnect	1062
33.4.8 fpgetpeername	1064
33.4.9 fpgetsockname	1064
33.4.10 fpgetsockopt	1065
33.4.11 fplisten	1065
33.4.12 fprecv	1065
33.4.13 fprecvfrom	1066
33.4.14 fpsend	1066
33.4.15 fsendto	1067
33.4.16 fpsetsockopt	1067
33.4.17 fpshutdown	1068
33.4.18 fpsocket	1068
33.4.19 fpsocketpair	1069
33.4.20 HostAddrToStr	1069
33.4.21 HostAddrToStr6	1069
33.4.22 HostToNet	1069
33.4.23 htonl	1070
33.4.24 hton	1070

33.4.25	NetAddrToStr	1070
33.4.26	NetAddrToStr6	1070
33.4.27	NetToHost	1071
33.4.28	NToHl	1071
33.4.29	NToHs	1071
33.4.30	ShortHostToNet	1071
33.4.31	ShortNetToHost	1072
33.4.32	Sock2File	1072
33.4.33	Sock2Text	1072
33.4.34	socketerror	1072
33.4.35	Str2UnixSockAddr	1073
33.4.36	StrToHostAddr	1073
33.4.37	StrToHostAddr6	1073
33.4.38	StrToNetAddr	1073
33.4.39	StrToNetAddr6	1074
34	Reference for unit 'strings'	1075
34.1	Overview	1075
34.2	Procedures and functions	1075
34.2.1	stralloc	1075
34.2.2	strcat	1075
34.2.3	strcmp	1076
34.2.4	strcpy	1076
34.2.5	strdispose	1077
34.2.6	strecopy	1077
34.2.7	strend	1078
34.2.8	stricmp	1079
34.2.9	stripos	1079
34.2.10	striscan	1080
34.2.11	strlcat	1080
34.2.12	strlcomp	1081
34.2.13	strlcopy	1081
34.2.14	strlen	1082
34.2.15	strlicomp	1082
34.2.16	strlower	1083
34.2.17	strmove	1083
34.2.18	strnew	1084
34.2.19	strpas	1085
34.2.20	strpcopy	1085
34.2.21	strpos	1086

34.2.22	strriscan	1086
34.2.23	strrscan	1087
34.2.24	strscan	1087
34.2.25	strupper	1087
35	Reference for unit 'strutils'	1088
35.1	Used units	1088
35.2	Constants, types and variables	1088
35.2.1	Resource strings	1088
35.2.2	Constants	1088
35.2.3	Types	1089
35.3	Procedures and functions	1090
35.3.1	AddChar	1090
35.3.2	AddCharR	1090
35.3.3	AnsiContainsStr	1091
35.3.4	AnsiContainsText	1091
35.3.5	AnsiEndsStr	1091
35.3.6	AnsiEndsText	1091
35.3.7	AnsiIndexStr	1092
35.3.8	AnsiIndexText	1092
35.3.9	AnsiLeftStr	1092
35.3.10	AnsiMatchStr	1093
35.3.11	AnsiMatchText	1093
35.3.12	AnsiMidStr	1093
35.3.13	AnsiProperCase	1093
35.3.14	AnsiReplaceStr	1094
35.3.15	AnsiReplaceText	1094
35.3.16	AnsiResemblesText	1094
35.3.17	AnsiReverseString	1094
35.3.18	AnsiRightStr	1095
35.3.19	AnsiStartsStr	1095
35.3.20	AnsiStartsText	1095
35.3.21	BinToHex	1095
35.3.22	Copy2Space	1096
35.3.23	Copy2SpaceDel	1096
35.3.24	Copy2Symb	1096
35.3.25	Copy2SymbDel	1097
35.3.26	Dec2Numb	1097
35.3.27	DecodeSoundexInt	1097
35.3.28	DecodeSoundexWord	1098

35.3.29	DelChars	1098
35.3.30	DelSpace	1098
35.3.31	DelSpace1	1098
35.3.32	DupeString	1099
35.3.33	ExtractDelimited	1099
35.3.34	ExtractSubstr	1099
35.3.35	ExtractWord	1100
35.3.36	ExtractWordPos	1100
35.3.37	FindPart	1100
35.3.38	GetCmdLineArg	1101
35.3.39	Hex2Dec	1101
35.3.40	HexToBin	1102
35.3.41	IfThen	1102
35.3.42	IntToBin	1102
35.3.43	IntToRoman	1103
35.3.44	IsEmptyStr	1103
35.3.45	IsWild	1103
35.3.46	IsWordPresent	1104
35.3.47	LeftBStr	1104
35.3.48	LeftStr	1104
35.3.49	MidBStr	1105
35.3.50	MidStr	1105
35.3.51	NPos	1105
35.3.52	Numb2Dec	1106
35.3.53	Numb2USA	1106
35.3.54	PadCenter	1106
35.3.55	PadLeft	1106
35.3.56	PadRight	1107
35.3.57	PosEx	1107
35.3.58	PosSet	1107
35.3.59	PosSetEx	1107
35.3.60	RandomFrom	1108
35.3.61	Removeleadingchars	1108
35.3.62	RemovePadChars	1108
35.3.63	RemoveTrailingChars	1109
35.3.64	ReplaceStr	1109
35.3.65	ReplaceText	1109
35.3.66	ReverseString	1109
35.3.67	RightBStr	1110
35.3.68	RightStr	1110

35.3.69	RomanToInt	1110
35.3.70	RomanToIntDef	1111
35.3.71	RPos	1111
35.3.72	RPosex	1111
35.3.73	SearchBuf	1111
35.3.74	Soundex	1112
35.3.75	SoundexCompare	1112
35.3.76	SoundexInt	1113
35.3.77	SoundexProc	1113
35.3.78	SoundexSimilar	1113
35.3.79	SoundexWord	1114
35.3.80	StringsReplace	1114
35.3.81	StuffString	1114
35.3.82	Tab2Space	1115
35.3.83	TrimLeftSet	1115
35.3.84	TrimRightSet	1115
35.3.85	TrimSet	1115
35.3.86	TryRomanToInt	1116
35.3.87	WordCount	1116
35.3.88	WordPosition	1116
35.3.89	XorDecode	1116
35.3.90	XorEncode	1117
35.3.91	XorString	1117
36	Reference for unit 'System'	1118
36.1	Overview	1118
36.2	Unicode and codepage support	1118
36.3	Miscellaneous functions	1119
36.4	Operating System functions	1120
36.5	String handling	1120
36.6	Mathematical routines	1121
36.7	Memory management functions	1122
36.8	File handling functions	1122
36.9	Run-Time Error behaviour	1123
36.10	Constants, types and variables	1123
36.10.1	Constants	1123
36.10.2	Types	1148
36.10.3	Variables	1179
36.11	Procedures and functions	1184
36.11.1	abs	1184

36.11.2	AbstractError	1184
36.11.3	AcquireExceptionObject	1184
36.11.4	AddExitProc	1185
36.11.5	Addr	1185
36.11.6	Align	1186
36.11.7	AllocMem	1186
36.11.8	AnsiToUtf8	1186
36.11.9	Append	1186
36.11.10	arctan	1187
36.11.11	ArrayStringToPPchar	1187
36.11.12	Assert	1188
36.11.13	Assign	1188
36.11.14	Assigned	1189
36.11.15	BasicEventCreate	1190
36.11.16	basiceventdestroy	1190
36.11.17	basiceventResetEvent	1190
36.11.18	basiceventSetEvent	1190
36.11.19	basiceventWaitFor	1190
36.11.20	BeginThread	1191
36.11.21	BEtoN	1191
36.11.22	binStr	1191
36.11.23	BlockRead	1192
36.11.24	BlockWrite	1193
36.11.25	Break	1193
36.11.26	BsfByte	1194
36.11.27	BsfDWord	1195
36.11.28	BsfQWord	1195
36.11.29	BsfWord	1195
36.11.30	BsrByte	1195
36.11.31	BsrDWord	1196
36.11.32	BsrQWord	1196
36.11.33	BsrWord	1196
36.11.34	CaptureBacktrace	1196
36.11.35	chdir	1197
36.11.36	chr	1197
36.11.37	Close	1198
36.11.38	CloseThread	1198
36.11.39	CompareByte	1198
36.11.40	CompareChar	1199
36.11.41	CompareChar0	1201

36.11.42 CompareDWord	1201
36.11.43 CompareWord	1202
36.11.44 Concat	1203
36.11.45 Continue	1204
36.11.46 Copy	1205
36.11.47 CopyArray	1205
36.11.48 cos	1206
36.11.49 Cseg	1206
36.11.50 Dec	1207
36.11.51 DefaultAnsi2UnicodeMove	1207
36.11.52 DefaultAnsi2WideMove	1208
36.11.53 DefaultUnicode2AnsiMove	1208
36.11.54 Delete	1208
36.11.55 Dispose	1209
36.11.56 DoneCriticalsection	1210
36.11.57 DoneThread	1210
36.11.58 Dseg	1210
36.11.59 DumpExceptionBackTrace	1211
36.11.60 Dump_Stack	1211
36.11.61 DynArrayBounds	1211
36.11.62 DynArrayClear	1212
36.11.63 DynArrayDim	1212
36.11.64 DynArrayIndex	1212
36.11.65 DynArraySetLength	1212
36.11.66 DynArraySize	1213
36.11.67 EmptyMethod	1213
36.11.68 EndThread	1213
36.11.69 EnterCriticalsection	1214
36.11.70 EnumResourceLanguages	1214
36.11.71 EnumResourceNames	1215
36.11.72 EnumResourceTypes	1215
36.11.73 EOF	1215
36.11.74 EOLn	1216
36.11.75 Erase	1217
36.11.76 Error	1217
36.11.77 Exclude	1217
36.11.78 Exit	1219
36.11.79 exp	1220
36.11.80 FilePos	1220
36.11.81 FileSize	1221

36.11.82 FillByte	1222
36.11.83 FillChar	1222
36.11.84 FillDWord	1223
36.11.85 FillQWord	1224
36.11.86 FillWord	1224
36.11.87 FinalizeArray	1224
36.11.88 FindResource	1225
36.11.89 FindResourceEx	1225
36.11.90 float_raise	1226
36.11.91 Flush	1226
36.11.92 FlushThread	1227
36.11.93 FMADouble	1227
36.11.94 FMAExtended	1227
36.11.95 FMASingle	1227
36.11.96 fpc_dynarray_rangecheck	1227
36.11.97 fpc_SarInt64	1227
36.11.98 FPower10	1228
36.11.99 frac	1228
36.11.100Freemem	1228
36.11.101Freememory	1229
36.11.102FreeResource	1229
36.11.103GetCPUCount	1229
36.11.104GetCurrentThreadId	1230
36.11.105getdir	1230
36.11.106GetFPCHeapStatus	1230
36.11.107GetHeapStatus	1231
36.11.108GetMem	1231
36.11.109GetMemory	1231
36.11.110GetMemoryManager	1231
36.11.111GetProcessID	1232
36.11.112GetResourceManager	1232
36.11.113GetTextCodePage	1232
36.11.114GetThreadID	1232
36.11.115GetThreadManager	1233
36.11.116GetUnicodeStringManager	1233
36.11.117GetVariantManager	1233
36.11.118GetWideStringManager	1233
36.11.119get_caller_addr	1234
36.11.120get_caller_frame	1234
36.11.121get_caller_stackinfo	1234

36.11.122get_cmdline	1235
36.11.123get_frame	1235
36.11.124Get_pc_addr	1235
36.11.125halt	1235
36.11.126hexStr	1236
36.11.127hi	1236
36.11.128High	1237
36.11.129HINSTANCE	1238
36.11.130Inc	1238
36.11.131Include	1239
36.11.132IndexByte	1240
36.11.133IndexChar	1240
36.11.134IndexChar0	1241
36.11.135IndexDWord	1241
36.11.136IndexQWord	1242
36.11.137Indexword	1242
36.11.138InitCriticalSection	1243
36.11.139InitializeArray	1244
36.11.140InitThread	1244
36.11.141InitThreadVars	1244
36.11.142Insert	1244
36.11.143int	1245
36.11.144InterlockedCompareExchange	1245
36.11.145InterLockedDecrement	1246
36.11.146InterLockedExchange	1246
36.11.147InterLockedExchangeAdd	1247
36.11.148InterLockedIncrement	1247
36.11.149IOResult	1247
36.11.150IsDynArrayRectangular	1249
36.11.151IsMemoryManagerSet	1249
36.11.152Is_IntResource	1249
36.11.153KillThread	1249
36.11.154LeaveCriticalsection	1250
36.11.155Length	1250
36.11.156LEtoN	1251
36.11.157ln	1251
36.11.158lo	1252
36.11.159LoadResource	1252
36.11.160LockResource	1253
36.11.161longjmp	1253

36.11.162Low	1253
36.11.163lowerCase	1254
36.11.164MakeLangID	1254
36.11.165MemSize	1254
36.11.166mkdir	1255
36.11.167Move	1255
36.11.168MoveChar0	1256
36.11.169New	1256
36.11.170NtoBE	1257
36.11.171NtoLE	1257
36.11.172Null	1257
36.11.173OctStr	1257
36.11.174odd	1258
36.11.175Ofs	1258
36.11.176operator *(variant, variant): variant	1259
36.11.177operator **(variant, variant): variant	1259
36.11.178operator +(variant, variant): variant	1260
36.11.179operator -(variant): variant	1260
36.11.180operator -(variant, variant): variant	1260
36.11.181operator /(variant, variant): variant	1261
36.11.182operator :=(ansistring): olevariant	1261
36.11.183operator :=(ansistring): variant	1261
36.11.184operator :=(Boolean): olevariant	1261
36.11.185operator :=(Boolean): variant	1261
36.11.186operator :=(Byte): olevariant	1262
36.11.187operator :=(Byte): variant	1262
36.11.188operator :=(Char): olevariant	1262
36.11.189operator :=(Char): variant	1262
36.11.190operator :=(comp): olevariant	1262
36.11.191operator :=(comp): variant	1262
36.11.192operator :=(currency): olevariant	1263
36.11.193operator :=(currency): variant	1263
36.11.194operator :=(Double): olevariant	1263
36.11.195operator :=(Double): variant	1263
36.11.196operator :=(DWord): olevariant	1263
36.11.197operator :=(DWord): variant	1264
36.11.198operator :=(extended): olevariant	1264
36.11.199operator :=(extended): variant	1264
36.11.200operator :=(Int64): olevariant	1264
36.11.201operator :=(Int64): variant	1264

36.11.202operator :=(longbool): olevariant	1264
36.11.203operator :=(longbool): variant	1265
36.11.204operator :=(LongInt): olevariant	1265
36.11.205operator :=(LongInt): variant	1265
36.11.206operator :=(olevariant): ansistring	1265
36.11.207operator :=(olevariant): Boolean	1265
36.11.208operator :=(olevariant): Byte	1266
36.11.209operator :=(olevariant): Char	1266
36.11.210operator :=(olevariant): comp	1266
36.11.211operator :=(olevariant): currency	1266
36.11.212operator :=(olevariant): Double	1266
36.11.213operator :=(olevariant): DWord	1266
36.11.214operator :=(olevariant): extended	1267
36.11.215operator :=(olevariant): Int64	1267
36.11.216operator :=(olevariant): longbool	1267
36.11.217operator :=(olevariant): LongInt	1267
36.11.218operator :=(olevariant): QWord	1267
36.11.219operator :=(olevariant): Real	1267
36.11.220operator :=(olevariant): ShortInt	1268
36.11.221operator :=(olevariant): shortstring	1268
36.11.222operator :=(olevariant): single	1268
36.11.223operator :=(olevariant): SmallInt	1268
36.11.224operator :=(olevariant): TDateTime	1268
36.11.225operator :=(olevariant): TError	1269
36.11.226operator :=(olevariant): UnicodeString	1269
36.11.227operator :=(olevariant): variant	1269
36.11.228operator :=(olevariant): WideChar	1269
36.11.229operator :=(olevariant): widestring	1269
36.11.230operator :=(olevariant): Word	1270
36.11.231operator :=(olevariant): wordbool	1270
36.11.232operator :=(QWord): olevariant	1270
36.11.233operator :=(QWord): variant	1270
36.11.234operator :=(Real): olevariant	1270
36.11.235operator :=(Real): variant	1270
36.11.236operator :=(real48): Double	1271
36.11.237operator :=(real48): extended	1271
36.11.238operator :=(ShortInt): olevariant	1271
36.11.239operator :=(ShortInt): variant	1271
36.11.240operator :=(shortstring): olevariant	1271
36.11.241operator :=(shortstring): variant	1271

36.11.242operator :=(single): olevariant	1272
36.11.243operator :=(single): variant	1272
36.11.244operator :=(SmallInt): olevariant	1272
36.11.245operator :=(SmallInt): variant	1272
36.11.246operator :=(TDateTime): olevariant	1272
36.11.247operator :=(TDateTime): variant	1272
36.11.248operator :=(TError): olevariant	1273
36.11.249operator :=(TError): variant	1273
36.11.250operator :=(UCS4String): variant	1273
36.11.251operator :=(UnicodeString): olevariant	1273
36.11.252operator :=(UnicodeString): variant	1273
36.11.253operator :=(UTF8String): variant	1273
36.11.254operator :=(variant): ansistring	1274
36.11.255operator :=(variant): Boolean	1274
36.11.256operator :=(variant): Byte	1274
36.11.257operator :=(variant): Char	1274
36.11.258operator :=(variant): comp	1274
36.11.259operator :=(variant): currency	1274
36.11.260operator :=(variant): Double	1275
36.11.261operator :=(variant): DWord	1275
36.11.262operator :=(variant): extended	1275
36.11.263operator :=(variant): Int64	1275
36.11.264operator :=(variant): longbool	1275
36.11.265operator :=(variant): LongInt	1275
36.11.266operator :=(variant): olevariant	1276
36.11.267operator :=(variant): QWord	1276
36.11.268operator :=(variant): Real	1276
36.11.269operator :=(variant): ShortInt	1276
36.11.270operator :=(variant): shortstring	1276
36.11.271operator :=(variant): single	1276
36.11.272operator :=(variant): SmallInt	1277
36.11.273operator :=(variant): TDateTime	1277
36.11.274operator :=(variant): TError	1277
36.11.275operator :=(variant): unicodestring	1277
36.11.276operator :=(variant): UTF8String	1277
36.11.277operator :=(variant): WideChar	1277
36.11.278operator :=(variant): widestring	1278
36.11.279operator :=(variant): Word	1278
36.11.280operator :=(variant): wordbool	1278
36.11.281operator :=(WideChar): olevariant	1278

36.11.282operator :=(WideChar): variant	1278
36.11.283operator :=(widestring): olevariant	1279
36.11.284operator :=(widestring): variant	1279
36.11.285operator :=(Word): olevariant	1279
36.11.286operator :=(Word): variant	1279
36.11.287operator :=(wordbool): olevariant	1279
36.11.288operator :=(wordbool): variant	1280
36.11.289operator <(variant, variant): Boolean	1280
36.11.290operator <=(variant, variant): Boolean	1280
36.11.291operator =(variant, variant): Boolean	1280
36.11.292operator >(variant, variant): Boolean	1281
36.11.293operator >=(variant, variant): Boolean	1281
36.11.294operator and(variant, variant): variant	1281
36.11.295operator div(variant, variant): variant	1282
36.11.296operator mod(variant, variant): variant	1282
36.11.297operator not(variant): variant	1282
36.11.298operator or(variant, variant): variant	1283
36.11.299operator shl(variant, variant): variant	1283
36.11.300operator shr(variant, variant): variant	1283
36.11.301operator xor(variant, variant): variant	1284
36.11.302Ord	1284
36.11.303Pack	1285
36.11.304Paramcount	1285
36.11.305ParamStr	1285
36.11.306pi	1286
36.11.307PopCnt	1286
36.11.308Pos	1287
36.11.309Power	1288
36.11.310Pred	1288
36.11.311prefetch	1289
36.11.312ptr	1289
36.11.313RaiseList	1289
36.11.314Random	1290
36.11.315Randomize	1290
36.11.316Read	1291
36.11.317ReadBarrier	1291
36.11.318ReadDependencyBarrier	1292
36.11.319ReadLn	1292
36.11.320ReadStr	1292
36.11.321ReadWriteBarrier	1293

36.11.322Real2Double	1293
36.11.323ReAllocMem	1294
36.11.324ReAllocMemory	1294
36.11.325ReleaseExceptionObject	1294
36.11.326Rename	1295
36.11.327Reset	1295
36.11.328ResumeThread	1296
36.11.329Rewrite	1296
36.11.330rmdir	1297
36.11.331RolByte	1298
36.11.332RolDWord	1298
36.11.333RolQWord	1299
36.11.334RolWord	1299
36.11.335RorByte	1299
36.11.336RorDWord	1299
36.11.337RorQWord	1300
36.11.338RorWord	1300
36.11.339round	1300
36.11.340RTLEventCreate	1301
36.11.341RTLeventdestroy	1301
36.11.342RTLeventResetEvent	1301
36.11.343RTLeventSetEvent	1302
36.11.344RTLeventWaitFor	1302
36.11.345RunError	1302
36.11.346SarInt64	1303
36.11.347SarLongint	1303
36.11.348SarShortint	1303
36.11.349SarSmallint	1303
36.11.350Seek	1304
36.11.351SeekEOF	1304
36.11.352SeekEOLn	1305
36.11.353Seg	1306
36.11.354SemaphoreDestroy	1306
36.11.355SemaphoreInit	1307
36.11.356SemaphorePost	1307
36.11.357SemaphoreWait	1307
36.11.358SetCodePage	1308
36.11.359Setjmp	1308
36.11.360SetLength	1309
36.11.361SetMemoryManager	1309

36.11.362SetMultiByteConversionCodePage	1309
36.11.363SetMultiByteFileSystemCodePage	1310
36.11.364SetMultiByteRTLFileSystemCodePage	1310
36.11.365SetResourceManager	1311
36.11.366SetString	1311
36.11.367SetTextBuf	1311
36.11.368SetTextCodePage	1312
36.11.369SetTextLineEnding	1313
36.11.370SetThreadManager	1313
36.11.371SetUnicodeStringManager	1313
36.11.372SetVariantManager	1314
36.11.373SetWideStringManager	1314
36.11.374ShortCompareText	1314
36.11.375sin	1315
36.11.376SizeOf	1315
36.11.377SizeofResource	1316
36.11.378Slice	1316
36.11.379Space	1316
36.11.380Sptr	1317
36.11.381sqr	1317
36.11.382sqrt	1318
36.11.383Sseg	1318
36.11.384StackTop	1318
36.11.385Str	1319
36.11.386StringCodePage	1319
36.11.387StringElementSize	1320
36.11.388StringOfChar	1320
36.11.389StringRefCount	1321
36.11.390StringToPPChar	1321
36.11.391StringToUnicodeChar	1321
36.11.392StringToWideChar	1322
36.11.393strlen	1322
36.11.394strpas	1322
36.11.395Succ	1322
36.11.396SuspendThread	1323
36.11.397Swap	1323
36.11.398SwapEndian	1324
36.11.399SysAllocMem	1324
36.11.400SysAssert	1324
36.11.401SysBackTraceStr	1324

36.11.402SysFlushStdIO	1325
36.11.403SysFreemem	1325
36.11.404SysFreememSize	1325
36.11.405SysGetFPCHeapStatus	1325
36.11.406SysGetHeapStatus	1325
36.11.407SysGetmem	1326
36.11.408SysInitExceptions	1326
36.11.409SysInitFPU	1326
36.11.410SysInitStdIO	1326
36.11.411SysMemSize	1326
36.11.412SysReAllocMem	1327
36.11.413SysResetFPU	1327
36.11.414SysSetCtrlBreakHandler	1327
36.11.415SysTryResizeMem	1327
36.11.416ThreadGetPriority	1328
36.11.417ThreadSetPriority	1328
36.11.418ThreadSwitch	1328
36.11.419ToSingleByteFileSystemEncodedFileName	1328
36.11.420trunc	1329
36.11.421Truncate	1329
36.11.422TryEnterCriticalsection	1330
36.11.423TypeInfo	1330
36.11.424UCS4StringToUnicodeString	1330
36.11.425UCS4StringToWideString	1331
36.11.426Unassigned	1331
36.11.427UnicodeCharLenToString	1331
36.11.428UnicodeCharLenToStrVar	1331
36.11.429UnicodeCharToString	1332
36.11.430UnicodeCharToStrVar	1332
36.11.431UnicodeStringToUCS4String	1332
36.11.432UnicodeToUtf8	1332
36.11.433UniqueString	1333
36.11.434UnlockResource	1333
36.11.435UnPack	1333
36.11.436upCase	1333
36.11.437Utf8CodePointLen	1334
36.11.438UTF8Decode	1335
36.11.439UTF8Encode	1335
36.11.440Utf8ToAnsi	1335
36.11.441Utf8ToUnicode	1336

36.11.442Val	1336
36.11.443VarArrayGet	1337
36.11.444VarArrayPut	1337
36.11.445VarArrayRedim	1337
36.11.446VarCast	1338
36.11.447WaitForThreadTerminate	1338
36.11.448WideCharLenToString	1338
36.11.449WideCharLenToStrVar	1338
36.11.450WideCharToString	1339
36.11.451WideCharToStrVar	1339
36.11.452WideStringToUCS4String	1339
36.11.453Write	1340
36.11.454WriteBarrier	1340
36.11.455WriteLn	1340
36.11.456WriteStr	1341
36.12 IDispatch	1342
36.12.1 Description	1342
36.12.2 Method overview	1342
36.12.3 IDispatch.GetTypeInfoCount	1342
36.12.4 IDispatch.GetTypeInfo	1342
36.12.5 IDispatch.GetIDsOfNames	1342
36.12.6 IDispatch.Invoke	1343
36.13 IEnumerable	1343
36.13.1 Description	1343
36.13.2 Method overview	1343
36.13.3 IEnumerable.GetEnumerator	1343
36.14 IEnumerator	1343
36.14.1 Description	1343
36.14.2 Method overview	1344
36.14.3 Property overview	1344
36.14.4 IEnumerator.GetCurrent	1344
36.14.5 IEnumerator.MoveNext	1344
36.14.6 IEnumerator.Reset	1345
36.14.7 IEnumerator.Current	1345
36.15 IInvokable	1345
36.15.1 Description	1345
36.16 IUnknown	1345
36.16.1 Description	1345
36.16.2 Method overview	1346
36.16.3 IUnknown.QueryInterface	1346

36.16.4	IUnknown._AddRef	1346
36.16.5	IUnknown._Release	1346
36.17	TAggregatedObject	1346
36.17.1	Description	1346
36.17.2	Method overview	1346
36.17.3	Property overview	1347
36.17.4	TAggregatedObject.Create	1347
36.17.5	TAggregatedObject.Controller	1347
36.18	TContainedObject	1347
36.18.1	Description	1347
36.18.2	Interfaces overview	1347
36.19	TInterfacedObject	1348
36.19.1	Description	1348
36.19.2	Interfaces overview	1348
36.19.3	Method overview	1348
36.19.4	Property overview	1348
36.19.5	TInterfacedObject.AfterConstruction	1348
36.19.6	TInterfacedObject.BeforeDestruction	1348
36.19.7	TInterfacedObject.NewInstance	1349
36.19.8	TInterfacedObject.RefCount	1349
36.20	TObject	1349
36.20.1	Description	1349
36.20.2	Method overview	1350
36.20.3	TObject.Create	1350
36.20.4	TObject.Destroy	1351
36.20.5	TObject.newInstance	1351
36.20.6	TObject.FreeInstance	1351
36.20.7	TObject.SafeCallException	1351
36.20.8	TObject.DefaultHandler	1352
36.20.9	TObject.Free	1352
36.20.10	TObject.InitInstance	1352
36.20.11	TObject.CleanupInstance	1352
36.20.12	TObject.ClassType	1353
36.20.13	TObject.ClassInfo	1353
36.20.14	TObject.ClassName	1353
36.20.15	TObject.ClassNameIs	1353
36.20.16	TObject.ClassParent	1354
36.20.17	TObject.InstanceSize	1354
36.20.18	TObject.InheritsFrom	1354
36.20.19	TObject.StringMessageTable	1354

36.20.20	TObject.MethodAddress	1355
36.20.21	TObject.MethodName	1355
36.20.22	TObject.FieldAddress	1355
36.20.23	TObject.AfterConstruction	1355
36.20.24	TObject.BeforeDestruction	1356
36.20.25	TObject.DefaultHandlerStr	1356
36.20.26	TObject.Dispatch	1356
36.20.27	TObject.DispatchStr	1356
36.20.28	TObject.GetInterface	1357
36.20.29	TObject.GetInterfaceByStr	1357
36.20.30	TObject.GetInterfaceWeak	1357
36.20.31	TObject.GetInterfaceEntry	1358
36.20.32	TObject.GetInterfaceEntryByStr	1358
36.20.33	TObject.GetInterfaceTable	1358
36.20.34	TObject.UnitName	1358
36.20.35	TObject.Equals	1359
36.20.36	TObject.GetHashCode	1359
36.20.37	TObject.ToString	1359
37	Reference for unit 'sysutils'	1360
37.1	Used units	1360
37.2	Overview	1360
37.3	Localization support	1360
37.4	Unicode and codepage awareness	1361
37.5	Miscellaneous conversion routines	1362
37.6	Date/time routines	1363
37.7	FileName handling routines	1363
37.8	File input/output routines	1364
37.9	PChar related functions	1365
37.10	Date and time formatting characters	1366
37.11	Formatting strings	1367
37.12	String functions	1367
37.13	Constants, types and variables	1368
37.13.1	Constants	1368
37.13.2	Types	1375
37.13.3	Variables	1382
37.14	Procedures and functions	1387
37.14.1	AbandonSignalHandler	1387
37.14.2	Abort	1387
37.14.3	AddDisk	1387

37.14.4	AddTerminateProc	1388
37.14.5	AdjustLineBreaks	1388
37.14.6	AnsiCompareFileName	1388
37.14.7	AnsiCompareStr	1389
37.14.8	AnsiCompareText	1390
37.14.9	AnsiDequotedStr	1391
37.14.10	AnsiExtractQuotedStr	1391
37.14.11	AnsiLastChar	1392
37.14.12	AnsiLowerCase	1392
37.14.13	AnsiLowerCaseFileName	1393
37.14.14	AnsiPos	1393
37.14.15	AnsiQuotedStr	1393
37.14.16	AnsiSameStr	1394
37.14.17	AnsiSameText	1394
37.14.18	AnsiStrComp	1394
37.14.19	AnsiStrIComp	1395
37.14.20	AnsiStrLastChar	1396
37.14.21	AnsiStrLComp	1397
37.14.22	AnsiStrLIComp	1397
37.14.23	AnsiStrLower	1398
37.14.24	AnsiStrPos	1399
37.14.25	AnsiStrRScan	1399
37.14.26	AnsiStrScan	1399
37.14.27	AnsiStrUpper	1400
37.14.28	AnsiUpperCase	1400
37.14.29	AnsiUpperCaseFileName	1401
37.14.30	AppendStr	1401
37.14.31	ApplicationName	1402
37.14.32	AssignStr	1402
37.14.33	BCDToInt	1403
37.14.34	Beep	1403
37.14.35	BoolToStr	1403
37.14.36	BytesOf	1404
37.14.37	ByteToCharIndex	1404
37.14.38	ByteToCharLen	1404
37.14.39	ByteType	1404
37.14.40	CallTerminateProcs	1405
37.14.41	ChangeFileExt	1405
37.14.42	CharInSet	1405
37.14.43	CharToByteLen	1406

37.14.44	CodePageNameToCodePage	1406
37.14.45	CodePageToCodePageName	1406
37.14.46	CompareMem	1406
37.14.47	CompareMemRange	1407
37.14.48	CompareStr	1407
37.14.49	CompareText	1408
37.14.50	ComposeDateTime	1409
37.14.51	ConcatPaths	1409
37.14.52	CreateDir	1410
37.14.53	CreateGUID	1410
37.14.54	CurrentYear	1411
37.14.55	CurrToStr	1411
37.14.56	CurrToStrF	1411
37.14.57	Date	1412
37.14.58	DateTimeToFileDate	1412
37.14.59	DateTimeToStr	1413
37.14.60	DateTimeToString	1413
37.14.61	DateTimeToSystemTime	1414
37.14.62	DateTimeToTimeStamp	1415
37.14.63	DateToStr	1415
37.14.64	DayOfWeek	1416
37.14.65	DecodeDate	1416
37.14.66	DecodeDateFully	1417
37.14.67	DecodeTime	1417
37.14.68	DeleteFile	1418
37.14.69	DirectoryExists	1418
37.14.70	DiskFree	1419
37.14.71	DiskSize	1419
37.14.72	DisposeStr	1420
37.14.73	DoDirSeparators	1420
37.14.74	EncodeDate	1421
37.14.75	EncodeTime	1422
37.14.76	ExceptAddr	1422
37.14.77	ExceptFrameCount	1422
37.14.78	ExceptFrames	1423
37.14.79	ExceptionErrorMessage	1423
37.14.80	ExceptObject	1423
37.14.81	ExcludeLeadingPathDelimiter	1424
37.14.82	ExcludeTrailingBackslash	1424
37.14.83	ExcludeTrailingPathDelimiter	1424

37.14.84 ExecuteProcess	1425
37.14.85 ExeSearch	1425
37.14.86 ExpandFileName	1426
37.14.87 ExpandFileNameCase	1426
37.14.88 ExpandUNCFileName	1427
37.14.89 ExtractFileDir	1428
37.14.90 ExtractFileDrive	1428
37.14.91 ExtractFileExt	1429
37.14.92 ExtractFileName	1429
37.14.93 ExtractFilePath	1429
37.14.94 ExtractRelativepath	1430
37.14.95 ExtractShortPathName	1430
37.14.96 FileAge	1431
37.14.97 FileClose	1431
37.14.98 FileCreate	1432
37.14.99 FileDateToDateTime	1433
37.14.100FileExists	1433
37.14.101FileGetAttr	1434
37.14.102FileGetDate	1435
37.14.103FileIsReadOnly	1436
37.14.104FileOpen	1436
37.14.105FileRead	1437
37.14.106FileSearch	1437
37.14.107FileSeek	1438
37.14.108FileSetAttr	1439
37.14.109FileSetDate	1439
37.14.110FileTruncate	1440
37.14.111FileWrite	1440
37.14.112FindClose	1440
37.14.113FindCmdLineSwitch	1441
37.14.114FindFirst	1441
37.14.115FindNext	1442
37.14.116FloattoCurr	1443
37.14.117FloatToDateTime	1443
37.14.118FloatToDecimal	1443
37.14.119FloatToStr	1444
37.14.120FloatToStrF	1445
37.14.121FloatToText	1447
37.14.122FloatToTextFmt	1448
37.14.123FmtStr	1448

37.14.124ForceDirectories	1449
37.14.125Format	1449
37.14.126FormatBuf	1456
37.14.127FormatCurr	1456
37.14.128FormatDateTime	1457
37.14.129FormatFloat	1457
37.14.130FreeAndNil	1459
37.14.131GetAppConfigDir	1459
37.14.132GetAppConfigFile	1460
37.14.133GetCurrentDir	1460
37.14.134GetDirs	1461
37.14.135GetEnvironmentString	1462
37.14.136GetEnvironmentVariable	1462
37.14.137GetEnvironmentVariableCount	1462
37.14.138GetFileHandle	1463
37.14.139GetLastError	1463
37.14.140GetLocalTime	1463
37.14.141GetLocalTimeOffset	1464
37.14.142GetModuleName	1464
37.14.143GetTempDir	1464
37.14.144GetTempFileName	1465
37.14.145GetTickCount	1465
37.14.146GetTickCount64	1465
37.14.147GetUserDir	1466
37.14.148GuidCase	1466
37.14.149GUIDToString	1466
37.14.150HashName	1466
37.14.151HookSignal	1467
37.14.152IncAMonth	1467
37.14.153IncludeLeadingPathDelimiter	1467
37.14.154IncludeTrailingBackslash	1468
37.14.155IncludeTrailingPathDelimiter	1468
37.14.156IncMonth	1468
37.14.157InquireSignal	1469
37.14.158IntToHex	1469
37.14.159IntToStr	1470
37.14.160IsDelimiter	1471
37.14.161IsEqualGUID	1471
37.14.162IsLeapYear	1471
37.14.163IsPathDelimiter	1472

37.14.164IsValidIdent	1472
37.14.165LastDelimiter	1473
37.14.166LeftStr	1473
37.14.167LoadStr	1474
37.14.168LowerCase	1474
37.14.169MSecsToTimeStamp	1474
37.14.170NewStr	1475
37.14.171Now	1475
37.14.172OutOfMemoryError	1476
37.14.173QuotedStr	1476
37.14.174RaiseLastOSError	1477
37.14.175RemoveDir	1477
37.14.176RenameFile	1477
37.14.177ReplaceDate	1478
37.14.178ReplaceTime	1478
37.14.179RightStr	1478
37.14.180SafeLoadLibrary	1479
37.14.181SameFileName	1479
37.14.182SameText	1479
37.14.183SetCurrentDir	1480
37.14.184SetDirSeparators	1480
37.14.185ShowException	1480
37.14.186Sleep	1481
37.14.187SScanf	1481
37.14.188StrAlloc	1482
37.14.189StrBufSize	1482
37.14.190StrByteType	1483
37.14.191strcat	1483
37.14.192StrCharLength	1483
37.14.193strcomp	1484
37.14.194StrCopy	1484
37.14.195StrDispose	1485
37.14.196strecopy	1485
37.14.197strend	1485
37.14.198StrFmt	1486
37.14.199stricomp	1487
37.14.200StringReplace	1487
37.14.201StringToGUID	1488
37.14.202strlcat	1488
37.14.203strlcomp	1489

37.14.204StrLCopy	1489
37.14.205StrLen	1490
37.14.206StrLFmt	1490
37.14.207strlicomp	1491
37.14.208strlower	1491
37.14.209strmove	1492
37.14.210strnew	1493
37.14.211StrNextChar	1493
37.14.212StrPas	1493
37.14.213StrPCopy	1494
37.14.214StrPLCopy	1494
37.14.215strpos	1494
37.14.216strrscan	1495
37.14.217strscan	1495
37.14.218StrToBool	1495
37.14.219StrToBoolDef	1496
37.14.220StrToCurr	1496
37.14.221StrToCurrDef	1496
37.14.222StrToDate	1497
37.14.223StrToDateDef	1497
37.14.224StrToDateTime	1498
37.14.225StrToDateTimeDef	1499
37.14.226StrToFloat	1499
37.14.227StrToFloatDef	1500
37.14.228StrToInt	1500
37.14.229StrToInt64	1501
37.14.230StrToInt64Def	1502
37.14.231StrToIntDef	1502
37.14.232StrToQWord	1503
37.14.233StrToQWordDef	1503
37.14.234StrToTime	1503
37.14.235StrToTimeDef	1504
37.14.236strupper	1504
37.14.237Supports	1504
37.14.238SysErrorMessage	1505
37.14.239SystemTimeToDateTime	1505
37.14.240TextToFloat	1506
37.14.241Time	1507
37.14.242TimeStampToDateTime	1508
37.14.243TimeStampToMSecs	1508

37.14.244TimeToStr	1509
37.14.245Trim	1509
37.14.246TrimLeft	1510
37.14.247TrimRight	1510
37.14.248TryEncodeDate	1511
37.14.249TryEncodeTime	1511
37.14.250TryFloatToCurr	1512
37.14.251TryStringToGUID	1512
37.14.252TryStrToBool	1512
37.14.253TryStrToCurr	1513
37.14.254TryStrToDate	1513
37.14.255TryStrToDateTime	1513
37.14.256TryStrToFloat	1514
37.14.257TryStrToInt	1514
37.14.258TryStrToInt64	1515
37.14.259TryStrToQWord	1515
37.14.260TryStrToTime	1515
37.14.261UnhookSignal	1516
37.14.262UpperCase	1516
37.14.263VendorName	1516
37.14.264WideCompareStr	1517
37.14.265WideCompareText	1517
37.14.266WideFmtStr	1517
37.14.267WideFormat	1518
37.14.268WideFormatBuf	1518
37.14.269WideLowerCase	1518
37.14.270WideSameStr	1519
37.14.271WideSameText	1519
37.14.272WideUpperCase	1519
37.14.273WrapText	1519
37.15 EAbort	1520
37.15.1 Description	1520
37.16 EAbstractError	1520
37.16.1 Description	1520
37.17 EAccessViolation	1520
37.17.1 Description	1520
37.18 EArgumentException	1520
37.18.1 Description	1520
37.19 EArgumentOutOfRangeException	1520
37.19.1 Description	1520

37.20	EAssertionFailed	1521
37.20.1	Description	1521
37.21	EBusError	1521
37.21.1	Description	1521
37.22	EControlC	1521
37.22.1	Description	1521
37.23	EConvertError	1521
37.23.1	Description	1521
37.24	EDivByZero	1521
37.24.1	Description	1521
37.25	EExternal	1521
37.25.1	Description	1521
37.26	EExternalException	1521
37.26.1	Description	1521
37.27	EFormatError	1522
37.27.1	Description	1522
37.28	EHeapMemoryError	1522
37.28.1	Description	1522
37.28.2	Method overview	1522
37.28.3	EHeapMemoryError.FreeInstance	1522
37.29	EInOutError	1522
37.29.1	Description	1522
37.30	EInterror	1522
37.30.1	Description	1522
37.31	EIntfCastError	1523
37.31.1	Description	1523
37.32	EIntOverflow	1523
37.32.1	Description	1523
37.33	EInvalidCast	1523
37.33.1	Description	1523
37.34	EInvalidContainer	1523
37.34.1	Description	1523
37.35	EInvalidInsert	1523
37.35.1	Description	1523
37.36	EInvalidOp	1523
37.36.1	Description	1523
37.37	EInvalidPointer	1523
37.37.1	Description	1523
37.38	EMathError	1524
37.38.1	Description	1524

37.39	ENoConstructException	1524
37.39.1	Description	1524
37.40	ENoThreadSupport	1524
37.40.1	Description	1524
37.41	ENotImplemented	1524
37.41.1	Description	1524
37.42	ENoWideStringSupport	1524
37.42.1	Description	1524
37.43	EObjectCheck	1524
37.43.1	Description	1524
37.44	EOSError	1525
37.44.1	Description	1525
37.45	EOutOfMemory	1525
37.45.1	Description	1525
37.46	EOverflow	1525
37.46.1	Description	1525
37.47	EPackageError	1525
37.47.1	Description	1525
37.48	EPrivilege	1525
37.48.1	Description	1525
37.49	EPropReadOnly	1525
37.49.1	Description	1525
37.50	EPropWriteOnly	1526
37.50.1	Description	1526
37.51	ERangeError	1526
37.51.1	Description	1526
37.52	ESafecallException	1526
37.52.1	Description	1526
37.53	EStackOverflow	1526
37.53.1	Description	1526
37.54	EUnderflow	1526
37.54.1	Description	1526
37.55	EVariantError	1526
37.55.1	Description	1526
37.55.2	Method overview	1526
37.55.3	EVariantError.CreateCode	1527
37.56	Exception	1527
37.56.1	Description	1527
37.56.2	Method overview	1527
37.56.3	Property overview	1527

37.56.4	Exception.Create	1527
37.56.5	Exception.CreateFmt	1528
37.56.6	Exception.CreateRes	1528
37.56.7	Exception.CreateResFmt	1528
37.56.8	Exception.CreateHelp	1528
37.56.9	Exception.CreateFmtHelp	1529
37.56.10	Exception.CreateResHelp	1529
37.56.11	Exception.CreateResFmtHelp	1529
37.56.12	Exception.ToString	1529
37.56.13	Exception.HelpContext	1530
37.56.14	Exception.Message	1530
37.57	EZeroDivide	1530
37.57.1	Description	1530
37.58	IReadWriteSync	1530
37.58.1	Description	1530
37.59	TMultiReadExclusiveWriteSynchronizer	1530
37.59.1	Description	1530
37.59.2	Interfaces overview	1531
37.60	TSimpleRWSync	1531
37.60.1	Description	1531
37.60.2	Interfaces overview	1531
38	Reference for unit 'types'	1532
38.1	Overview	1532
38.2	Constants, types and variables	1532
38.2.1	Constants	1532
38.2.2	Types	1537
38.3	Procedures and functions	1543
38.3.1	Bounds	1543
38.3.2	CenterPoint	1543
38.3.3	EqualRect	1543
38.3.4	InflateRect	1544
38.3.5	IntersectRect	1544
38.3.6	IsRectEmpty	1544
38.3.7	OffsetRect	1544
38.3.8	Point	1545
38.3.9	PtInRect	1545
38.3.10	Rect	1545
38.3.11	Size	1545
38.3.12	UnionRect	1546

38.4	IClassFactory	1546
38.4.1	Description	1546
38.4.2	Method overview	1546
38.4.3	IClassFactory.CreateInstance	1546
38.4.4	IClassFactory.LockServer	1546
38.5	ISequentialStream	1547
38.5.1	Description	1547
38.5.2	Method overview	1547
38.5.3	ISequentialStream.Read	1547
38.5.4	ISequentialStream.Write	1547
38.6	IStream	1547
38.6.1	Description	1547
38.6.2	Method overview	1548
38.6.3	IStream.Seek	1548
38.6.4	IStream.SetSize	1548
38.6.5	IStream.CopyTo	1548
38.6.6	IStream.Commit	1549
38.6.7	IStream.Revert	1549
38.6.8	IStream.LockRegion	1549
38.6.9	IStream.UnlockRegion	1549
38.6.10	IStream.Stat	1550
38.6.11	IStream.Clone	1550
39	Reference for unit 'typinfo'	1551
39.1	Used units	1551
39.2	Overview	1551
39.3	Auxiliary functions	1551
39.4	Getting or setting property values	1552
39.5	Examining published property information	1552
39.6	Constants, types and variables	1553
39.6.1	Constants	1553
39.6.2	Types	1554
39.7	Procedures and functions	1561
39.7.1	FindPropInfo	1561
39.7.2	GetEnumName	1562
39.7.3	GetEnumNameCount	1563
39.7.4	GetEnumProp	1563
39.7.5	GetEnumValue	1564
39.7.6	GetFloatProp	1564
39.7.7	GetInt64Prop	1565

39.7.8	GetInterfaceProp	1566
39.7.9	GetMethodProp	1566
39.7.10	GetObjectProp	1568
39.7.11	GetObjectPropClass	1569
39.7.12	GetOrdProp	1570
39.7.13	GetPropInfo	1571
39.7.14	GetPropInfos	1571
39.7.15	GetPropList	1572
39.7.16	GetPropValue	1573
39.7.17	GetRawInterfaceProp	1574
39.7.18	GetSetProp	1574
39.7.19	GetStrProp	1575
39.7.20	GetTypeData	1576
39.7.21	GetUnicodeStrProp	1577
39.7.22	GetVariantProp	1577
39.7.23	GetWideStrProp	1577
39.7.24	IsPublishedProp	1578
39.7.25	IsStoredProp	1578
39.7.26	PropIsType	1579
39.7.27	PropType	1580
39.7.28	SetEnumProp	1581
39.7.29	SetFloatProp	1581
39.7.30	SetInt64Prop	1582
39.7.31	SetInterfaceProp	1582
39.7.32	SetMethodProp	1583
39.7.33	SetObjectProp	1583
39.7.34	SetOrdProp	1584
39.7.35	SetPropValue	1584
39.7.36	SetRawInterfaceProp	1584
39.7.37	SetSetProp	1585
39.7.38	SetStrProp	1585
39.7.39	SetToString	1586
39.7.40	SetUnicodeStrProp	1587
39.7.41	SetVariantProp	1587
39.7.42	SetWideStrProp	1587
39.7.43	StringToSet	1588
39.8	EPropertyConvertError	1588
39.8.1	Description	1588
39.9	EPropertyError	1588
39.9.1	Description	1588

40 Reference for unit 'Unix'	1589
40.1 Used units	1589
40.2 Constants, types and variables	1589
40.2.1 Constants	1589
40.2.2 Types	1596
40.2.3 Variables	1605
40.3 Procedures and functions	1605
40.3.1 AssignPipe	1605
40.3.2 AssignStream	1606
40.3.3 FpExecL	1607
40.3.4 FpExecLE	1608
40.3.5 FpExecLP	1609
40.3.6 FpExecLPE	1610
40.3.7 FpExecV	1610
40.3.8 FpExecVP	1611
40.3.9 FpExecVPE	1612
40.3.10 fpFlock	1613
40.3.11 fpfStatFS	1613
40.3.12 fpfsync	1614
40.3.13 fpgettimeofday	1614
40.3.14 fpStatFS	1614
40.3.15 fpSystem	1615
40.3.16 FSearch	1615
40.3.17 GetDomainName	1616
40.3.18 GetHostName	1616
40.3.19 GetLocalTimezone	1617
40.3.20 GetTimezoneFile	1617
40.3.21 PClose	1618
40.3.22 POpen	1618
40.3.23 ReadTimezoneFile	1619
40.3.24 ReReadLocalTime	1619
40.3.25 SeekDir	1620
40.3.26 SelectText	1620
40.3.27 SigRaise	1620
40.3.28 TellDir	1621
40.3.29 WaitProcess	1621
40.3.30 WIFSTOPPED	1622
40.3.31 W_EXITCODE	1622
40.3.32 W_STOPCODE	1622

41 Reference for unit 'unixtype'	1623
41.1 Overview	1623
41.2 Constants, types and variables	1623
41.2.1 Constants	1623
41.2.2 Types	1625
42 Reference for unit 'unixutil'	1638
42.1 Overview	1638
42.2 Constants, types and variables	1638
42.2.1 Types	1638
42.2.2 Variables	1638
42.3 Procedures and functions	1639
42.3.1 ArrayStringToPPchar	1639
42.3.2 EpochToLocal	1639
42.3.3 GetFS	1640
42.3.4 GregorianToJulian	1640
42.3.5 JulianToGregorian	1641
42.3.6 LocalToEpoch	1641
42.3.7 StringToPPChar	1641
43 Reference for unit 'video'	1643
43.1 Overview	1643
43.2 Examples utility unit	1644
43.3 Writing a custom video driver	1644
43.4 Constants, types and variables	1648
43.4.1 Constants	1648
43.4.2 Types	1652
43.4.3 Variables	1654
43.5 Procedures and functions	1655
43.5.1 ClearScreen	1655
43.5.2 DefaultErrorHandler	1655
43.5.3 DoneVideo	1656
43.5.4 GetCapabilities	1656
43.5.5 GetCursorType	1657
43.5.6 GetLockScreenCount	1658
43.5.7 GetVideoDriver	1659
43.5.8 GetVideoMode	1659
43.5.9 GetVideoModeCount	1660
43.5.10 GetVideoModeData	1661
43.5.11 InitVideo	1661
43.5.12 LockScreenUpdate	1661

43.5.13	SetCursorPos	1662
43.5.14	SetCursorType	1663
43.5.15	SetVideoDriver	1663
43.5.16	SetVideoMode	1664
43.5.17	UnlockScreenUpdate	1664
43.5.18	UpdateScreen	1664
44	Reference for unit 'winert'	1666
44.1	Overview	1666
44.2	Constants, types and variables	1666
44.2.1	Variables	1666
44.3	Procedures and functions	1666
44.3.1	delay	1666
44.3.2	keypressed	1666
44.3.3	nosound	1667
44.3.4	readkey	1667
44.3.5	sound	1667
44.3.6	textmode	1667
45	Reference for unit 'x86'	1668
45.1	Used units	1668
45.2	Overview	1668
45.3	Procedures and functions	1668
45.3.1	fpIOperm	1668
45.3.2	fpIoPL	1669
45.3.3	ReadPort	1669
45.3.4	ReadPortB	1669
45.3.5	ReadPortL	1670
45.3.6	ReadPortW	1670
45.3.7	WritePort	1670
45.3.8	WritePortB	1671
45.3.9	WritePortl	1671
45.3.10	WritePortW	1671

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	1118
unixtype	1623

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.

`BITSINWORD = 8 * (cuLong)`

Number of bits in a word.

`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

ESysENOCSSI = 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

ESysENOPROTOOPT = 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` (135) 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` (149) command: Get close-on-exec flag

`F_GetFl = 3`

`fpFCntl` (149) command: Get filedescriptor flags

`F_GetLk = 5`

`fpFCntl` (149) command: Get lock

`F_GetOwn = 9`

fpFCntl (149) command: get owner of filedescriptor events

F_OK = 0

fpAccess (141) call test: file exists.

F_SetFd = 2

fpFCntl (149) command: Set close-on-exec flag

F_SetFl = 4

fpFCntl (149) command: Set filedescriptor flags

F_SetLk = 6

fpFCntl (149) command: Set lock

F_SetLkW = 7

fpFCntl (149) command: Test lock

F_SetOwn = 8

fpFCntl (149) command: Set owner of filedescriptor events

ln2bitmask = 1 shl ln2bitsinword - 1

Last bit in word.

ln2bitsinword = 5

Power of 2 number of bits in word.

MAP_ANON = MAP_ANONYMOUS

Anonymous memory mapping (data private to application)

MAP_ANONYMOUS = \$20

FpMMap (163) map type: Don't use a file

MAP_DENYWRITE = \$800

FpMMap (163) option: Ignored.

MAP_EXECUTABLE = \$1000

FpMMap (163) option: Ignored.

MAP_FAILED = (-1)

Memory mapping failed error code

MAP_FIXED = \$10

FpMMap (163) map type: Interpret addr exactly

MAP_GROWSDOWN = \$100

FpMMap (163) option: Memory grows downward (like a stack)

MAP_LOCKED = \$2000

FpMMap (163) option: lock the pages in memory.

MAP_NORESERVE = \$4000

FpMMap (163) option: Do not reserve swap pages for this memory.

MAP_PRIVATE = \$2

FpMMap (163) map type: Changes are private

MAP_SHARED = \$1

FpMMap (163) map type: Share changes

MAP_TYPE = \$f

FpMMap (163) map type: Bitmask for type of mapping

NAME_MAX = UnixType . NAME_MAX

Maximum filename length.

O_APPEND = \$400

fpOpen (167) file open mode: Append to file

O_CREAT = \$40

fpOpen (167) file open mode: Create if file does not yet exist.

O_DIRECT = \$4000

fpOpen (167) file open mode: Minimize caching effects

O_DIRECTORY = \$10000

fpOpen (167) file open mode: File must be directory.

O_EXCL = \$80

`fpOpen (167)` file open mode: Open exclusively

`O_NDELAY = O_NONBLOCK`

`fpOpen (167)` file open mode: Alias for `O_NonBlock (114)`

`O_NOCTTY = $100`

`fpOpen (167)` file open mode: No TTY control.

`O_NOFOLLOW = $20000`

`fpOpen (167)` file open mode: Fail if file is symbolic link.

`O_NONBLOCK = $800`

`fpOpen (167)` file open mode: Open in non-blocking mode

`O_RDONLY = 0`

`fpOpen (167)` file open mode: Read only

`O_RDWR = 2`

`fpOpen (167)` file open mode: Read/Write

`O_SYNC = $1000`

`fpOpen (167)` file open mode: Write to disc at once

`O_TRUNC = $200`

`fpOpen (167)` file open mode: Truncate file to length 0

`O_WRONLY = 1`

`fpOpen (167)` 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 ([1623](#))

PRIO_PROCESS = UnixType . PRIO_PROCESS

Easy access alias for unixtype.PRIO_PROCESS ([1623](#))

PRIO_USER = UnixType . PRIO_USER

Easy access alias for unixtype.PRIO_USER ([1623](#))

PROT_EXEC = \$4

FpMMap ([163](#)) memory access: page can be executed

PROT_NONE = \$0

FpMMap ([163](#)) memory access: page can not be accessed

PROT_READ = \$1

FpMMap (163) memory access: page can be read

PROT_WRITE = 2

FpMMap (163) 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 filesize

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 ([141](#)) 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 ([179](#)) 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 ([161](#)) option: Set position relative to current position.

SEEK_END = 2

fpLSeek (161) option: Set position relative to end of file.

SEEK_SET = 0

fpLSeek (161) 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 (longint) - 3`

Signal information pad size.

`SYS_NMLN = UnixType . SYS_NMLN`

Max system name length.

`S_IFBLK = 24576`

File (#rtl.baseunix.stat (135) record) mode: Block device

`S_IFCHR = 8192`

File (#rtl.baseunix.stat (135) record) mode: Character device

`S_IFDIR = 16384`

File (#rtl.baseunix.stat (135) record) mode: Directory

`S_IFIFO = 4096`

File (#rtl.baseunix.stat (135) record) mode: FIFO

`S_IFLNK = 40960`

File (#rtl.baseunix.stat (135) record) mode: Link

`S_IFMT = 61440`

File (#rtl.baseunix.stat (135) record) mode: File type bit mask

`S_IFREG = 32768`

File (#rtl.baseunix.stat (135) record) mode: Regular file

`S_IFSOCK = 49152`

File (#rtl.baseunix.stat (135) record) mode: Socket

S_IRGRP = %0000100000

Mode flag: Read by group.

S_IROTH = %0000000100

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_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 (140) domain name.

UTSNAME_LENGTH = SYS_NMLN

Max length of utsname (140) system name, release, version, machine.

UTSNAME_NODENAME_LENGTH = UTSNAME_LENGTH

Max length of utsname (140) node name.

WNOHANG = 1

#rtl.baseunix.fpWaitpid (192) option: Do not wait for processes to terminate.

wordsinfdset = FD_MAXFDSET div BITSINWORD

Number of words in a TFDSet (135) array

wordsinsigset = SIG_MAXSIG div BITSINWORD

Number of words in a signal set.

WUNTRACED = 2

#rtl.baseunix.fpWaitpid (192) option: Also report children which were stopped but not yet reported

W_OK = 2

fpAccess (141) call test: write allowed

X_OK = 1

fpAccess (141) 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` ([1625](#))

`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` ([124](#)).

`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` ([125](#)).

`csshort = UnixType.csshort`

Short signed integer type

`cuchar = UnixType.cuchar`

Alias for `#rtl.UnixType.cuchar` ([1626](#))

`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 ([1627](#))

```
cushort = UnixType.cushort
```

C type: short unsigned integer (half sized)

```
dev_t = UnixType.dev_t
```

Device descriptor type.

```
Dir = record
  dd_fd : Integer;
  dd_loc : LongInt;
  dd_size : Integer;
  dd_buf : pDirent;
  dd_nextoff : Cardinal;
  dd_max : Integer;
  dd_lock : pointer;
end
```

Record used in fpOpenDir ([168](#)) and fpReadDir ([172](#)) calls

```
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` (172) function to return files in a directory.

```
FLock = record
  l_type : cshort;
  l_whence : cshort;
  l_start : kernel_off_t;
  l_len : kernel_off_t;
  l_pid : pid_t;
end
```

Lock description type for `fpFCntl` (149) lock call.

```
gid_t = UnixType.gid_t
```

Group ID type.

```
ino_t = UnixType.ino_t
```

Inode type.

```
iovec = record
  iov_base : pointer;
  iov_len : size_t;
end
```

`iovec` is used in `freadv` (174) for IO to multiple buffers to describe a buffer location.

```
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 (135) type.

`PBlkSize = ^Blksize_t`

Pointer to TBlkSize (135) type.

`pcbool = UnixType.pcbbool`

Pointer to boolean type cbool (124)

`pcchar = UnixType.pcchar`

Alias for `#rtl.UnixType.pcchar` (1628)

`pcdouble = UnixType.pcdouble`

Pointer to cdouble (124) type.

`pcfloat = UnixType.pcfloating`

Pointer to cfloat (124) type.

`pcint = UnixType.pcint`

Pointer to cInt (124) 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` (135) type.

`pclong = UnixType.pclong`

Pointer to `cLong` (124) type.

`pclonglong = UnixType.pclonglong`

Pointer to `longlong` type.

`pcschar = UnixType.pcschar`

Pointer to character type `cschar` (125).

`pcshort = UnixType.pcsshort`

Pointer to `cShort` (125) type.

`pcsigned = UnixType.pcsigned`

Pointer to signed integer type `csigned` (125).

`pcsint = UnixType.pcsint`

Pointer to signed integer type `csint` (125)

`pcsize_t = UnixType.psize_t`

Pointer to `csize_t`

`pcslong = UnixType.pcslong`

Pointer of the signed long `cslong` (125)

`pcslonglong = UnixType.pcslonglong`

Pointer to Signed longlong type `cslonglong` (125)

`pcsshort = UnixType.pcsshort`

Pointer to short signed integer type `csshort` (125)

`pcuchar = UnixType.pcuchar`

Alias for `#rtl.UnixType.pcuchar` (1630)

`pcuint = UnixType.pcuint`

Pointer to cUInt (125) 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 (126) type.

```
pculonglong = UnixType.pculonglong
```

Unsigned longlong type

```
pcunsigned = UnixType.pcsunsigned
```

Alias for #rtl.unixtype.pcsunsigned (1630)

```
pcushort = UnixType.pcushort
```

Pointer to cuShort (126) type.

```
pDev = UnixType.pDev
```

Pointer to TDev (135) type.

```
pDir = ^Dir
```

Pointer to TDir (135) record

```
pDirent = ^Dirent
```

Pointer to TDirent (135) record.

```
pFDSet = ^TFDSet
```

Pointer to TFDSet (135) type.

```
pFilDes = ^TFilDes
```

Pointer to TFileDes (135) type.

```
pfpstate = ^tfpstate
```

Pointer to tfpstate (136) record.

```
pGid = UnixType.pGid
```

Pointer to TGid (136) type.

```
pGrpArr = ^TGrpArr
```

Pointer to TGrpArr (136) array.

```
pid_t = UnixType.pid_t
```

Process ID type.

```
pIno = UnixType.pIno
```

Pointer to TIno (136) type.

```
piovec = ^tiovec
```

pointer to a iovec (127) record

```
pMode = UnixType.pMode
```

Pointer to TMode (137) type.

```
pnLink = UnixType.pnLink
```

Pointer to TnLink (137) type.

```
pOff = UnixType.pOff
```

Pointer to TOff (137) type.

```
pollfd = record
  fd : cint;
  events : cshort;
  revents : cshort;
end
```

pollfd is used in the fpPoll (170) call to describe the various actions.

```
pPid = UnixType.pPid
```

Pointer to TPid (137) type.

`ppollfd = ^pollfd`

Pointer to `tpollfd`.

`PRLimit = ^TRLimit`

Pointer to `TRLimit` (137) record

`psigactionrec = ^sigactionrec`

Pointer to `SigActionRec` (134) record type.

`PSigContext = ^TSigContext`

Pointer to `#rtl.baseunix.TSigContext` (138) record type.

`psiginfo = ^tsiginfo`

Pointer to `#rtl.baseunix.TSigInfo` (138) record type.

`psigset = ^tsigset`

Pointer to `SigSet` (134) type.

`pSize = UnixType.pSize`

Pointer to `TSize` (139) type.

`pSize_t = UnixType.pSize_t`

Pointer to `Size_t`

`pSocklen = UnixType.pSocklen`

Pointer to `TSockLen` (139) type.

`psSize = UnixType.psSize`

Pointer to `TsSize` (139) type

`PStat = ^Stat`

Pointer to `TStat` (139) type.

`pstatfs = UnixType.PStatFs`

This is an alias for the type defined in the `#rtl.unixtype` (1623) 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 (139) type.

`ptimespec = UnixType.ptimespec`

Pointer to timespec (136) type.

`ptimeval = UnixType.ptimeval`

Pointer to timeval (136) type.

`ptimezone = ^timezone`

Pointer to TimeZone (136) record.

`ptime_t = UnixType.ptime_t`

Pointer to time_t (136) type.

`PTms = ^tms`

Pointer to TTms (139) type.

`Pucontext = ^Tucontext`

Pointer to TUContext (140) type.

`pUId = UnixType.pUId`

Pointer to TUID (140) type.

`pUtimBuf = ^UtimBuf`

Pointer to TUTimBuf (140) type.

`PUtsName = ^TUTsName`

Pointer to TUTsName (140) type.

`rlim_t = culong`

`rlim_t` is used as the type for the various fields in the TRLimit (137) 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

```
sigactionrec = record  
  sa_handler : sigactionhandler_t;  
  sa_flags : culong;  
  sa_restorer : sigrestorerhandler_t;  
  sa_mask : sigset_t;  
end
```

Record used in `fpSigAction` (179) call.

`signalhandler = signalhandler_t`

Simple signal handler prototype

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

Standard signal handler prototype

```
sigrestorerhandler = sigrestorerhandler_t
```

Alias for `sigrestorerhandler_t` (134) 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.

```
Stat = packed record
end
```

Record describing an inode (file) in the FPFstat (152) call.

```
TBlkCnt = Blkcnt_t
```

Alias for Blkcnt_t (124) type.

```
TBlkSize = Blksize_t
```

Alias for blksize_t (124) type.

```
TClock = UnixType.TClock
```

Alias for clock_t (124) type.

```
TDev = UnixType.TDev
```

Alias for dev_t (126) type.

```
TDir = Dir
```

Alias for Dir (126) type.

```
TDirent = Dirent
```

Alias for Dirent (127) type.

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

File descriptor set for fpSelect (175) call.

```
TFilDes = Array[0..1] of cint
```

Array of file descriptors as used in fpPipe (169) call.

```
tfpreg = record
  significand : Array[0..3] of Word;
  exponent : Word;
end
```

Record describing floating point register in signal handler.

```
tfpstate = record
  cw : Cardinal;
  sw : Cardinal;
  tag : Cardinal;
  ipoff : Cardinal;
  cssel : Cardinal;
  dataoff : Cardinal;
  datasel : Cardinal;
  st : Array[0..7] of tfpreg;
  status : Cardinal;
end
```

Record describing floating point unit in signal handler.

```
TGid = UnixType.TGid
```

Alias for `gid_t` (127) type.

```
TGrpArr = Array[0..0] of TGid
```

Array of `gid_t` (127) IDs

```
timespec = UnixType.timespec
```

Short time specification type.

```
timeval = UnixType.timeval
```

Time specification type.

```
timezone = record
  tz_minuteswest : cint;
  tz_dsttime : cint;
end
```

Record describing a timezone

```
time_t = UnixType.time_t
```

Time span type

```
TIno = UnixType.TIno
```

Alias for `ino_t` (127) type.

```
TIOCtlRequest = UnixType.TIOCtlRequest
```

Easy access alias for `unixtype.TIOCtlRequest` (1635)

```
tiovec = iovec
```

Alias for the `iovec` (127) record type.

```
TMode = UnixType.TMode
```

Alias for `mode_t` (128) type.

```
tms = record
  tms_utime : clock_t;
  tms_stime : clock_t;
  tms_cutime : clock_t;
  tms_cstime : clock_t;
end
```

Record containing timings for `fpTimes` (189) call.

```
TnLink = UnixType.TnLink
```

Alias for `nlink_t` (128) type.

```
TOff = UnixType.TOff
```

Alias for `off_t` (128) type.

```
TPid = UnixType.TPid
```

Alias for `pid_t` (131) type.

```
tpollfd = pollfd
```

Alias for `pollfd` type

```
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.

```
tsigactionhandler = sigactionhandler_t
```

Alias for `sigactionhandler_t` (134) type.

```
tsigaltstack = record
  ss_sp : pointer;
  ss_flags : LongInt;
  ss_size : LongInt;
end
```

Provide the location of an alternate signal handler stack.

```

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.

```

tsiginfo = record
  si_signo : LongInt;
  si_errno : LongInt;
  si_code : LongInt;
  _sifields : record
  end;
end

```

This type describes the signal that occurred.

```
tsignalhandler = signalhandler_t
```

Alias for `signalhandler_t` (134) type.

```
tsigrestorerhandler = sigrestorerhandler_t
```

Alias for `sigrestorerhandler_t` (134) type.

`tsigset = sigset_t`

Alias for `SigSet` (134) type.

`TSize = UnixType.TSize`

Alias for `size_t` (134) type

`TSocklen = UnixType.TSocklen`

Alias for `socklen_t` (134) type.

`TsSize = UnixType.TsSize`

Alias for `ssize_t` (135) type

`TStat = Stat`

Alias for `Stat` (135) type.

`tstatfs = UnixType.TStatFs`

Record describing a file system in the `unix.fstatfs` (100) call.

`TTime = UnixType.TTime`

Alias for `TTime` (139) type.

`Ttimespec = UnixType.Ttimespec`

Alias for `TimeSpec` (136) type.

`TTimeVal = UnixType.TTimeVal`

Alias for `timeval` (136) type.

`TTimeZone = timezone`

Alias for `TimeZone` (136) record.

`TTms = tms`

Alias for `Tms` (137) record type.

```
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.

```
TUId = UnixType.TUId
```

Alias for uid_t (140) type.

```
TUtimBuf = UtimBuf
```

Alias for UtimBuf (140) type.

```
TUtsName = UtsName
```

Alias for UtsName (140) type.

```
uid_t = UnixType.uid_t
```

User ID type

```
UtimBuf = record
  actime : time_t;
  modtime : time_t;
end
```

Record used in fpUtime (190) to set file access and modification times.

```
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.

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: Ansistring) : 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 (147) and friends.

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

See also: fpExecv (147), FreeShellArgV (193)

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_OKUser has read rights.

W_OKUser has write rights.

X_OKUser has execute rights.

F_OKFile 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` (154).

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

sys_einvalMode was incorrect.

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

sys_enotdirA directory component in `Path` is not a directory.

sys_enomemInsufficient kernel memory.

sys_eloop`Path` has a circular symbolic link.

See also: `FpChown` (144), `FpChmod` (142)

Listing: `./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` (179), `fpPause` (169)

Listing: `./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 succesful, -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` (154).

See also: `fpGetCwd` (153)

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_ISUIDSet user ID on execution.
S_ISGIDSet Group ID on execution.
S_ISVTXSet sticky bit.
S_IRUSRRead by owner.
S_IWUSRWrite by owner.
S_IXUSRExecute by owner.
S_IRGRPRead by group.
S_IWGRPWrite by group.
S_IXGRPExecute by group.
S_IROTHRead by others.
S_IWOTHWrite by others.
S_IXOTHExecute by others.
S_IRWXORead, write, execute by others.
S_IRWXGRead, write, execute by groups.
S_IRWXURead, 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_epermThe effective UID doesn't match the ownership of the file, and is not zero. Owner or group were not specified correctly.
sys_eaccessOne of the directories in `Path` has no search (=execute) permission.
sys_enoentA directory entry in `Path` does not exist or is a symbolic link pointing to a non-existent directory.
sys_enomemInsufficient kernel memory.
sys_erofsThe file is on a read-only filesystem.
sys_eLOOP`Path` has a reference to a circular symbolic link, i.e. a symbolic link, whose expansion points to itself.

See also: `fpChown` ([144](#)), `fpAccess` ([141](#))

Listing: `./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);
WriteIn (f, '#!/bin/sh');
WriteIn (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 succesfull, 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 filesystem.

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

See also: `fpChmod` ([142](#)), `fpAccess` ([141](#))

Listing: `./bunixex/ex24.pp`

Program Example24;

{ Program to demonstrate the Chown function. }

Uses BaseUnix;

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

begin

```

WriteIn ('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` (167).

Errors: Extended error information can be retrieved using `fpGetErrno` (154).

See also: `FpOpen` (167), `FpRead` (171), `FpWrite` (192), `FpFTruncate` (153), `FpLSeek` (161)

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` (168).

Errors: Extended error information can be retrieved using `fpGetErrno` (154).

See also: `FpOpenDir` (168), `FpReadDir` (172)

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 `filides`.

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 fileposition in both files.

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

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` ([146](#))

Listing: `./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 filehandle 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 fileposition in both files.

The function returns the new file descriptor number, on error -1 is returned, and the error can be retrieved with `fpgeterrno` (154)

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

sys_ebadfOldFile (or `filides`) hasn't been assigned.

sys_emfileMaximum number of open files for the process is reached.

See also: `fpDup` (145)

Listing: `./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 `argvp`. 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` (154)

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

sys_epermThe file system is mounted `\textit{noexec}`.

sys_e2bigArgument list too big.

sys_enoexecThe magic number in the file is incorrect.

sys_enoentThe file does not exist.

sys_enomemNot enough memory for kernel.

sys_enotdirA component of the path is not a directory.

sys_eloopThe path contains a circular reference (via symlinks).

See also: [fpExecve \(148\)](#), [fpFork \(151\)](#)

Listing: ./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 \(154\)](#), and includes the following:

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

sys_epermThe file system is mounted \textit{noexec}.

sys_e2bigArgument list too big.

sys_enoexecThe magic number in the file is incorrect.

sys_enoentThe file does not exist.

sys_enomemNot 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` ([147](#)), `fpFork` ([151](#))

Listing: `./bunixex/ex7.pp`

Program `Example7`;

{ Program to demonstrate the Execve function. }

Uses `BaseUnix`, `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 }
{ Env is defined in system.inc }
fpExecVe ('/bin/lS',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` ([151](#)), `fpExecve` ([148](#))

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. `fd` is a valid file descriptor. `cmd` specifies what to do, and is one of the following:

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

F_GetFlRead the descriptor's flags.

F_GetOwnGet the Process ID of the owner of a socket.

F_SetFdSet the `close_on_exec` flag of `fd`. (only the least significant bit is used).

F_GetLkReturn the `flock` record that prevents this process from obtaining the lock, or set the `l_type` field of the lock if there is no obstruction. `arg` is the `flock` record.

F_SetLkSet 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_GetLkwSame as for **F_Setlk**, but wait until the lock is released.

F_SetOwnSet 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` (154) for extended error information.

sys_ebadf`fd` has a bad file descriptor.

sys_eagain or sys_eaccessFor `\textbf{F_SetLk}`, if the lock is held by another process.

1.4.15 `fpfdfillset`

Synopsis: Set all filedescriptors in the set.

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

Visibility: default

Description: `fpfdfillset` sets all filedescriptors in `nset`.

See also: `FpSelect` (175), `FpFD_ZERO` (151), `FpFD_IsSet` (151), `FpFD_Clr` (150), `FpFD_Set` (151)

1.4.16 `fpFD_CLR`

Synopsis: Clears a filedescriptor in a set

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

Visibility: default

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

For an example, see `FpSelect` (175).

Errors: None.

See also: `FpSelect` (175), `FpFD_ZERO` (151), `FpFD_Set` (151), `FpFD_IsSet` (151)

1.4.17 fpFD_ISSET

Synopsis: Check whether a filedescriptor is set

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

Visibility: default

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

For an example, see `FpSelect` (175).

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

See also: `FpSelect` (175), `FpFD_ZERO` (151), `FpFD_Clr` (150), `FpFD_Set` (151)

1.4.18 fpFD_SET

Synopsis: Set a filedescriptor in a set

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

Visibility: default

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

For an example, see `FpSelect` (175).

Errors: None.

See also: `FpSelect` (175), `FpFD_ZERO` (151), `FpFD_Clr` (150), `FpFD_IsSet` (151)

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 filedescriptors in the file descriptor set `nset`.

For an example, see `FpSelect` (175).

Errors: None.

See also: `FpSelect` (175), `FpFD_Clr` (150), `FpFD_Set` (151), `FpFD_IsSet` (151)

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` (157)).

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 \(148\)](#), [#rtl.linux.Clone \(763\)](#)

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` ([135](#)). The function returns zero if the call was successful, a nonzero return value indicates failure.

Errors: Extended error information can be retrieved using `fpGetErrno` ([154](#)).

sys_enoent Path does not exist.

See also: [FpStat \(184\)](#), [FpLStat \(161\)](#)

Listing: `./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` ([154](#)).

See also: `FpOpen` ([167](#)), `FpClose` ([145](#)), `FpRead` ([171](#)), `FpWrite` ([192](#)), `FpLSeek` ([161](#))

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 succesful, 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` ([156](#)), `FpGetUID` ([158](#))

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` (155), `FpGetUid` (158), `FpGetEUid` (155), `FpGetPid` (156), `FpGetPPid` (157), `fpSetUID` (178), `FpSetGid` (177)

Listing: `./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 accomodate for strings longer than 255 bytes, `TERMCAP` and `LS_COLORS`, for instance.

Errors: None.

Listing: `./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` (177)

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` (158), `FpGetGid` (155), `FpGetEGid` (153), `FpGetPid` (156), `FpGetPPid` (157), `fpSetUID` (178), `FpSetGid` (177)

Listing: `./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` (153), `FpGetUid` (158), `FpGetEUid` (155), `FpGetPid` (156), `FpGetPPid` (157), `fpSetUID` (178), `FpSetGid` (177)

Listing: `./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 ([154](#))

See also: FpGetpgrp ([156](#)), FpGetGID ([155](#)), FpGetEGID ([153](#))

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 ([156](#)), fpGetPPID ([157](#)), FpGetGID ([155](#)), FpGetUID ([158](#))

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 ([157](#))

Listing: ./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` ([156](#))

Listing: `./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 pre-defined `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` ([166](#)).

Errors: Error information is returned solely by the `FpGetErrno` ([154](#)) function: a priority can be a positive or negative value.

sys_esrchNo process found using `which` and `who`.

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

See also: `FpSetPriority` ([177](#)), `FpNice` ([166](#))

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` (137) 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` (154) can be used to retrieve the error code.

See also: `FpSetRLimit` (178)

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: `FpGetpgrp` (156), `FpGetpid` (156), `FpGetPpid` (157)

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` (155), `FpGetEUid` (155), `FpGetEGid` (153), `FpGetPid` (156), `FpGetPPid` (157), `fpSetUID` (178)

Listing: `./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 filedescriptor `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` ([154](#)).

Listing: `./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` ([154](#)):

sys_einvalAn invalid signal is sent.

sys_esrchThe `Pid` or process group don't exist.

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

See also: `FpSigAction` ([179](#)), `FpSignal` ([181](#))

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 als `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 succesfull, 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 filesystem.
sys_eperm The filesystem 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 filesystem.
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` (185), `fpUnLink` (190)

Listing: `./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 FPUntlink ( '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 fileposition of file `fd` to `Offset`, starting from `Whence`, which can be one of the following:

`Seek_SetOffset` is the absolute position in the file.

`Seek_CurOffset` is relative to the current position.

`Seek_endOffset` is relative to the end of the file.

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

For an example, see `FpOpen` ([167](#)).

Errors: Extended error information can be retrieved using `fpGetErrno` ([154](#)).

See also: `FpOpen` ([167](#)), `FpWrite` ([192](#)), `FpClose` ([145](#)), `FpRead` ([171](#)), `FpFTruncate` ([153](#))

1.4.41 fpLstat

Synopsis: Return information about symbolic link. Do not 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` ([152](#)), it stores information about the link, not about the file the link points to. The function returns zero if the call was succesful, a nonzero return value indicates failure. failed.

Errors: Extended error information is returned by the `FpGetErrno` ([154](#)) function.

`sys_enoent``Path` does not exist.

See also: [FpFStat \(152\)](#), [#rtl.unixtype.TStatFS \(1636\)](#)

Listing: ./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 succes, 0 is returned. if the function fails, -1 is returned.

Note: There exist a portable alterative 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` ([154](#)).

See also: `fpGetCWD` ([153](#)), `fpChDir` ([142](#))

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 filesystem, with name `Path` and mode `Mode`.

The function returns zero if the command was succesful, 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:

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

lenSize (in bytes) of area to be mapped.

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

PROT_EXECThe memory can be executed.

PROT_READThe memory can be read.

PROT_WRITEThe memory can be written.

PROT_NONEThe memory can not be accessed.

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

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

MAP_SHAREDShare this map with other processes that map this object.

MAP_PRIVATECreate 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.

fdFile descriptor from which to map.

offOffset 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` (154) 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_EINVALOne 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_ENOMEMNot enough memory for this operation.

See also: `FpMUnMap` (165)

Listing: `./unixex/ex66.pp`

Program `Example66`;

{ Program to demonstrate the MMap function. }

Uses `BaseUnix`, `Unix`;

```
Var S      : String;
    fd     : cint;
    Len    : longint;
//    args : tmmargs;
    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(fpmmap(nil,len+1,PROT_READ or PROT_WRITE,MAP_PRIVATE,fd,0));

If longint(P)=-1 then
  Halt(4);
WriteIn('Read in memory :',P);
fpclose(fd);
if fpMUnMap(P,Len)<>0 Then
  Halt(fpgeterrno);
end.

```

1.4.45 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` (163).

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

For an example, see `FpMMap` (163).

Errors: In case of error the function returns a nonzero value, extended error information is returned by the `FpGetErrno` (154) function. See `FpMMap` (163) for possible error values.

See also: `FpMMap` (163)

1.4.46 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` (154).

See also: `FpPause` (169), `FpAlarm` (141)

Listing: ./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;
end.

```

1.4.47 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 succesful, zero is returned. On error, a nonzero value is returned.

Errors: Extended error information is returned by the `FpGetErrno` ([154](#)) function.

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

See also: `FpGetPriority` ([157](#)), `FpSetPriority` ([177](#))

Listing: ./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.

```

1.4.48 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_RdOnlyFile is opened Read-only

O_WrOnlyFile is opened Write-only

O_RdWrFile is opened Read-Write

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

O_CreatFile is created if it doesn't exist.

O_ExclIf the file is opened with **O_Creat** and it already exists, the call will fail.

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

O_TruncIf the file exists, it will be truncated.

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

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

O_NDelayIdem as **O_NonBlock**

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

O_NoFollowif the file is a symbolic link, the open fails. (linux 2.1.126 and higher only)

O_Directoryif 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 filedescriptor, or a negative value if there was an error.

Errors: Extended error information can be retrieved using fpGetErrno ([154](#)).

See also: [FpClose \(145\)](#), [FpRead \(171\)](#), [FpWrite \(192\)](#), [FpFTruncate \(153\)](#), [FpLSeek \(161\)](#)

Listing: ./bunixex/ex19.pp

Program Example19;

{ Program to demonstrate the fdOpen, fdwrite and fdClose 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.

1.4.49 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 ([126](#)) 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 ([154](#)).

See also: [FpCloseDir \(145\)](#), [FpReadDir \(172\)](#)

Listing: ./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 : ',d_off);
            Writeln ( 'Reclen   : ',d_reclen);
            Writeln ( 'Name     : ',pchar(@d_name[0]));
          end;
        end;
      Until Entry=-1;
      fpCloseDir ( TheDir^);
    end.

```

1.4.50 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 sigal, the handler will be called and after that pause will return control to the process.

For an example, see fpAlarm ([141](#)).

1.4.51 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 filehandles 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 succesfully, 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` ([1618](#)), `fpMkFifo` ([163](#))

Listing: `./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.
```

1.4.52 FpPoll

Synopsis: Poll a file descriptor for events.

Declaration: `function FpPoll (fds : ppollfd ; nfd : 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 `nfd`. `>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` ([137](#))

1.4.53 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` (174), `FpPWrite` (171)

1.4.54 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` (170), `FpWriteV` (193)

1.4.55 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` (154).

See also: `FpOpen` (167), `FpClose` (145), `FpWrite` (192), `FpFTruncate` (153), `FpLSeek` (161)

Listing: `./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;
                end;
            end;
        end;
    end.

```

1.4.56 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` (127) record describing the entry. If the next entry can't be read, `Nil` is returned.

For an example, see [FpOpenDir \(168\)](#).

Errors: Extended error information can be retrieved using [fpGetErrno \(154\)](#).

See also: [FpCloseDir \(145\)](#), [FpOpenDir \(168\)](#)

1.4.57 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 \(154\)](#) function.

SYS_ENOTDIRA 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_ENAMETOOLONGA pathname, or a component of a pathname, was too long.

SYS_ENOENTthe link name does not exist.

SYS_EACCESNo permission to search a directory in the path

SYS_ELOOPToo many symbolic links were encountered in translating the pathname.

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

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

SYS_ENOMEMNot enough kernel memory was available.

See also: [FpSymLink \(185\)](#)

Listing: `./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.

```

1.4.58 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` (137) buffers pointed to by `iov`. It works like `fpRead` (171) only on multiple buffers.

Errors: On error, -1 is returned.

See also: `FpWriteV` (193), `FpPWrite` (171), `FpPRead` (170)

1.4.59 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` (154).

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

sys_enotdirpart of `Old` or `NewPath` is not directory.

sys_efaultFor 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_enosp the filesystem has no room for the new directory entry.

See also: [FpUnLink \(190\)](#)

1.4.60 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 \(154\)](#).

1.4.61 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 \(151\)](#), [fpFD_Clr \(150\)](#), [fpFD_Set \(151\)](#) or [fpFD_IsSet \(151\)](#) 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` (154).

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

SYS_EINTRA 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` (151), `fpFD_Clr` (150), `fpFD_Set` (151), `fpFD_IsSet` (151)

Listing: `./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.

1.4.62 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` ([154](#))

1.4.63 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` ([154](#)).

See also: `FpSetUid` ([178](#)), `FpGetGid` ([155](#)), `FpGetUid` ([158](#)), `FpGetEUid` ([155](#)), `FpGetEGid` ([153](#)), `FpGetPid` ([156](#)), `FpGetPPid` ([157](#))

1.4.64 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 pre-defined 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` ([166](#)).

The function returns zero on success, -1 on failure

Errors: Extended error information is returned by the `FpGetErrno` ([154](#)) function.

sys_esrchNo process found using `which` and `who`.

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

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

sys_eaccessA non-superuser tried to a priority increase.

See also: `FpGetPriority` ([157](#)), `FpNice` ([166](#))

1.4.65 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` (137) 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` (154) can be used to retrieve the error code.

See also: `FpGetRLimit` (157)

1.4.66 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 `FpGetpgrp` (156)). 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` (154)

1.4.67 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` (1614)

1.4.68 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` (154).

See also: `FpGetGid` (155), `FpGetUid` (158), `FpGetEUid` (155), `FpGetEGid` (153), `FpGetPid` (156), `FpGetPPid` (157), `FpSetGid` (177)

1.4.69 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` (134) 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_NOCLDSTOPIf `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_RESTARTFor 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` (154).

sys_einvalan invalid signal was specified, or it was **SIGKILL** or **SIGSTOP**.

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

sys_eintrSystem call was interrupted.

See also: `FpSigProcMask` (182), `FpSigPending` (182), `FpSigSuspend` (183), `FpKill` (159)

Listing: `./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.

```

1.4.70 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` (180), `FpSigFillSet` (181), `FpSigDelSet` (180), `FpSigIsMember` (181)

1.4.71 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` (180), `FpSigFillSet` (181), `FpSigAddSet` (180), `FpSigIsMember` (181)

1.4.72 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` (181), `FpSigAddSet` (180), `FpSigDelSet` (180), `FpSigIsMember` (181)

1.4.73 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` (180), `FpSigAddSet` (180), `FpSigDelSet` (180), `FpSigIsMember` (181)

1.4.74 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` (180), `FpSigFillSet` (181), `FpSigAddSet` (180), `FpSigDelSet` (180)

1.4.75 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` (179) call. The return value for `FpSignal` is the old signal handler, or nil on error.

Errors: Extended error information can be retrieved using `fpGetErrno` (154).

SIG_ERR An error occurred.

See also: `FpSigAction` (179), `FpKill` (159)

Listing: `./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( SigUsrc1 , SignalHandler (@DoSig))= signalhandler (SIG_ERR) then
        begin
            writeln( 'Error: ',fpGetErrno , '. ');
            halt(1);
        end;
        Writeln ( 'Send USR1 signal or press <ENTER> to exit ');
        readln;
end.

```

1.4.76 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` (179), `fpSigProcMask` (182), `fpSigSuspend` (183), `fpSignal` (181), `fpKill` (159)

1.4.77 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_BLOCKThe set of blocked signals is the union of the current set and the `nset` argument.

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

SIG_SETMASKThe 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 adress outside the range of the process.

sys_eintrSystem call was interrupted.

See also: `fpSigAction` (179), `fpSigPending` (182), `fpSigSuspend` (183), `fpKill` (159)

1.4.78 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` (179), `FpSigProcMask` (182), `FpSigPending` (182), `FpSignal` (181), `FpKill` (159)

1.4.79 FpSigTimedWait

Synopsis: Wait for signal, with timeout

Declaration: `function FpSigTimedWait(const sigset: tsigset; info: psigninfo; 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` (183)

1.4.80 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` (169), `FpAlarm` (141), `FpNanoSleep` (165)

Listing: `./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.

```

1.4.81 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` (135). The function returns zero if the call was succesfull, a nonzero return value indicates failure.

Errors: Extended error information can be retrieved using `fpGetErrno` (154).

`sys_enoent``Path` does not exist.

See also: `FpStat` (184), `FpLStat` (161)

Listing: `./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.

```

1.4.82 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 succesful, a nonzero value if the call failed.

Errors: Extended error information is returned by the `FpGetErrno` (154) function.

sys_epermThe filesystem containing oldpath and newpath does not support linking files.

sys_eaccessWrite access for the directory containing Newpath is disallowed, or one of the directories in OldPath or NewPath has no search (=execute) permission.

sys_enoentA directory entry in OldPath or NewPath does not exist or is a symbolic link pointing to a non-existent directory.

sys_enotdirA directory entry in OldPath or NewPath is nor a directory.

sys_enomemInsufficient kernel memory.

sys_erofsThe files are on a read-only filesystem.

sys_eexistNewPath already exists.

sys_eloopOldPath or NewPath has a reference to a circular symbolic link, i.e. a symbolic link, whose expansion points to itself.

sys_enospcThe device containing NewPath has no room for another entry.

See also: `FpLink` (160), `FpUnLink` (190), `FpReadLink` (173)

Listing: ./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.
```

1.4.83 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 ([152](#)), FpS_ISLNK ([187](#)), FpS_ISREG ([188](#)), FpS_ISDIR ([187](#)), FpS_ISCHR ([186](#)), FpS_ISFIFO ([187](#)), FpS_ISSOCK ([188](#))

1.4.84 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 ([152](#)), FpS_ISLNK ([187](#)), FpS_ISREG ([188](#)), FpS_ISDIR ([187](#)), FpS_ISBLK ([186](#)), FpS_ISFIFO ([187](#)), FpS_ISSOCK ([188](#))

1.4.85 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` (152), `FpS_ISLNK` (187), `FpS_ISREG` (188), `FpS_ISCHR` (186), `FpS_ISBLK` (186), `fpS_ISFIFO` (187), `FpS_ISSOCK` (188)

1.4.86 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` (152), `FpS_ISLNK` (187), `FpS_ISREG` (188), `FpS_ISCHR` (186), `FpS_ISBLK` (186), `FpS_ISDIR` (187), `FpS_ISSOCK` (188)

1.4.87 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` (152), `FpS_ISFIFO` (187), `FpS_ISREG` (188), `FpS_ISCHR` (186), `FpS_ISBLK` (186), `FpS_ISDIR` (187), `FpS_ISSOCK` (188)

Listing: `./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.

```

1.4.88 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` (152), `fpS_ISFIFO` (187), `fpS_ISLNK` (187), `fpS_ISCHR` (186), `fpS_ISBLK` (186), `fpS_ISDIR` (187), `fpS_ISSOCK` (188)

1.4.89 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` (152), `fpS_ISFIFO` (187), `fpS_ISLNK` (187), `fpS_ISCHR` (186), `fpS_ISBLK` (186), `fpS_ISDIR` (187), `fpS_ISREG` (188)

1.4.90 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 00:00:00 GMT, january 1, 1970. it is adjusted to the local time zone, but not to DST. 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` (154).

Listing: `./bunixex/ex1.pp`

Program Example1;

{ Program to demonstrate the fptime function. }

Uses baseunix;

begin

Write ('Secs past the start of the Epoch (00:00 1/1/1980) : ');

Writeln (fptime);

end.

1.4.91 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` ([154](#)).

See also: `fpUtime` ([190](#))

1.4.92 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` ([142](#))

Listing: `./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.

1.4.93 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` (154).

See also: `FpUTime` (190)

1.4.94 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 succesfull, 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` (160).

Errors: Extended error information can be retrieved using `fpGetErrno` (154).

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 filesystem.

See also: `FpLink` (160), `FpSymLink` (185)

1.4.95 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 filesystem (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` (154).

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` (188), `FpChown` (144), `FpAccess` (141)

Listing: `./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, ', ', ' ');
      Writeln ( Day:2, '/', month:2, '/', year:4 );
    end;
end.
```

1.4.96 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` (154).

See also: `fpFork` (151), `fpExecve` (148), `fpWaitPid` (192)

1.4.97 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` (763) 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` (151).

Errors: Extended error information can be retrieved using `fpgetErrno` (154).

See also: `fpFork` (151), `fpExecve` (148), `fpWait` (191)

1.4.98 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` (154).

See also: `FpOpen` (167), `FpClose` (145), `FpRead` (171), `FpFTruncate` (153), `FpLSeek` (161)

1.4.99 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 `iovec` (137) buffers pointed to by `iov`. It works like `fpWrite` (192) only from multiple buffers.

Errors: On error, -1 is returned.

See also: `FpReadV` (174), `FpPWrite` (171), `FpPRead` (170)

1.4.100 FreeShellArgV

Synopsis: Free the result of a `CreateShellArgV` (140) 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` (140).

Errors: None.

See also: `CreateShellArgV` (140)

1.4.101 wexitStatus

Synopsis: Extract the exit status from the `fpWaitPID` (192) 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` (192) call.

See also: `FpWaitPID` (192), `WTERMSIG` (194), `WSTOPSIG` (194), `WIFEXITED` (193), `WIFSIGNALED` (194)

1.4.102 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` (192), `WTERMSIG` (194), `WSTOPSIG` (194), `WIFSIGNALED` (194), `WEXITSTATUS` (193)

1.4.103 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` ([192](#)), `WTERMSIG` ([194](#)), `WSTOPSIG` ([194](#)), `WIFEXITED` ([193](#)), `WEXITSTATUS` ([193](#))

1.4.104 wstopsig

Synopsis: Return the exit code from the process.

Declaration: `function wstopsig(Status: cint) : cint`

Visibility: default

Description: `WSTOPSIG` is an alias for `WEXITSTATUS` ([193](#)).

See also: `FpWaitPID` ([192](#)), `WTERMSIG` ([194](#)), `WIFEXITED` ([193](#)), `WIFSIGNALED` ([194](#)), `WEXITSTATUS` ([193](#))

1.4.105 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` ([192](#)), `WSTOPSIG` ([194](#)), `WIFEXITED` ([193](#)), `WIFSIGNALED` ([194](#)), `WEXITSTATUS` ([193](#))

Chapter 2

Reference for unit 'Classes'

2.1 Used units

Table 2.1: Used units by unit 'Classes'

Name	Page
rtlconsts	??
System	1118
sysutils	1360
types	1532
typinfo	1551

2.2 Overview

This documentation describes the FPC `classes` unit. The `Classes` unit contains basic classes for the Free Component Library (FCL):

- a `TList` ([330](#)) class for maintaining lists of pointers,
- `TStringList` ([383](#)) for lists of strings,
- `TCollection` ([280](#)) to manage collections of objects
- `TStream` ([369](#)) classes to support streaming.

Furthermore it introduces methods for object persistence, and classes that understand an owner-owned relationship, with automatic memory management.

2.3 Constants, types and variables

2.3.1 Constants

`BITSHIFT` = 5

Used to calculate the size of a bits array

`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` (230) 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.

`fmCreate = $FF00`

`TFileStream.Create` (310) creates a new file if needed.

`fmOpenRead = 0`

`TFileStream.Create` (310) opens a file with read-only access.

`fmOpenReadWrite = 2`

`TFileStream.Create` (310) opens a file with read-write access.

`fmOpenWrite = 1`

`TFileStream.Create` (310) opens a file with write-only access.

`MASK = 31`

Bitmask with all bits on.

`MaxBitFlags = $7FFFFFFE0`

Maximum number of bits in `TBits` collection.

`MaxBitRec = MaxBitFlags div ((cardinal) * 8)`

Maximum number of bit records in `TBits`.

`MaxListSize = Maxint div 16`

This constant sets the maximum number of elements in a `TList` (330).

`scAlt = $8000`

Indicates ALT key in a keyboard shortcut.

```
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 (371) starts relative to the stream origin.

```
soFromCurrent = 1
```

Seek (371) starts relative to the current position in the stream.

```
soFromEnd = 2
```

Seek (371) starts relative to the stream end.

```
toEOF = (0)
```

Value returned by TParser.Token (349) when the end of the input stream was reached.

```
toFloat = (4)
```

Value returned by TParser.Token (349) when a floating point value was found in the input stream.

```
toInteger = (3)
```

Value returned by TParser.Token (349) when an integer was found in the input stream.

```
toString = (2)
```

Value returned by TParser.Token (349) when a string was found in the input stream.

```
toSymbol = (1)
```

Value returned by TParser.Token (349) when a symbol was found in the input stream.

```
toWString = (5)
```

Value returned by TParser.Token (349) when a widestring was found in the input stream.

2.3.2 Types

```
HMODULE = PtrInt
```

FPC doesn't support modules yet, so this is a dummy type.

```
HRSRC = TFPResourceHandle deprecated
```

This type is provided for Delphi compatibilty, it is used for resource streams.

```
PPointerList = ^TPointerList
```

Pointer to an array of pointers.

```
PStringItem = ^TStringItem
```

Pointer to a TStringItem (209) record.

```
PStringItemList = ^TStringItemList
```

Pointer to a TStringItemList (209).

```
TActiveXRegType = (axrComponentOnly, axrIncludeDescendants)
```

Table 2.2: Enumeration values for type TActiveXRegType

Value	Explanation
axrComponentOnly	
axrIncludeDescendants	

This type is provided for compatibility only, and is currently not used in Free Pascal.

```
TAlignment = (taLeftJustify, taRightJustify, taCenter)
```

Table 2.3: 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` (255) class reference.

`TBasicActionLinkClass = Class of TBasicActionLink`

`TBasicActionLink` (259) class reference.

`TBiDiMode = (bdLeftToRight, bdRightToLeft, bdRightToLeftNoAlign,
bdRightToLeftReadingOnly)`

Table 2.4: Enumeration values for type `TBiDiMode`

Value	Explanation
<code>bdLeftToRight</code>	Texts read from left to right.
<code>bdRightToLeft</code>	Texts read from right to left.
<code>bdRightToLeftNoAlign</code>	Texts read from right to left, but not right-aligned
<code>bdRightToLeftReadingOnly</code>	Texts read from right to left

`TBiDiMode` describes bi-directional support for displaying texts.

`TBitArray = Array[0..MaxBitRec-1] of Cardinal`

Array to store bits.

`TCollectionItemClass = Class of TCollectionItem`

`TCollectionItemClass` is used by the `TCollection.ItemClass` (286) property of `TCollection` (280) to identify the descendent class of `TCollectionItem` (288) which should be created and managed.

`TCollectionNotification = (cnAdded, cnExtracting, cnDeleting)`

Table 2.5: 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` (280) 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` (285) 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` (290) descendent 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 2.6: 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.
<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` (354) 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` (354) 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` (419) 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` (294)

```
TComponentStyle= Set of (csInheritable,csCheckPropAvail,csSubComponent,
                           csTransient)
```

Table 2.7: Enumeration values for type

Value	Explanation
<code>csCheckPropAvail</code>	??
<code>csInheritable</code>	The component can be on inherited forms.
<code>csSubComponent</code>	Subcomponent - streamed as part of the owning component
<code>csTransient</code>	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` (304).

```
TDuplicates = Types.TDuplicates
```

Type to describe what to do with duplicate values in a `TStringlist` (383).

`TExceptionClass = Class of Exception`

`TExceptionClass` is the class pointer for the `Exception` (1527) class, defined in the `SysUtils` (1360) unit.

`TFilerFlag = (ffInherited, ffChildPos, ffInline)`

Table 2.8: Enumeration values for type `TFilerFlag`

Value	Explanation
<code>ffChildPos</code>	The position of the child on it's parent is included.
<code>ffInherited</code>	Stored object is an inherited object.
<code>ffInline</code>	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` (202)

```
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` (215) 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` (354) instance needs to locate a method and it doesn't find it in the streamed form, then the `OnFindMethod` (364) 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 2.9: 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` (320) stream descendents

`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 2.10: 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.

```
TIdentMapEntry = record
  Value : Integer;
  Name : string;
end
```

TIdentMapEntry is used internally by the IdentToInt (217) and IntToIdent (218) calls to store the mapping between the identifiers and the integers they represent.

```
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 (223) call.

```
TInitComponentHandler = function(Instance: TComponent;
                                RootAncestor: TClass) : Boolean
```

TInitComponentHandler is a callback type. It is used in the InitInheritedComponent (218) call to initialize a component. Callbacks of this type are registered with the RegisterInitComponentHandler (223) call.

```
TIntToIdent = function(Int: LongInt;var Ident: string) : Boolean
```

TIntToIdent 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 (223) call.

```
TLeftRight = taLeftJustify..taRightJustify
```

TLeftRight is a subrange type based on the TAlignment (198) enumerated type. It contains only the left and right alignment constants.

```
TListAssignOp = (laCopy, laAnd, laOr, laXor, laSrcUnique, laDestUnique)
```

Table 2.11: Enumeration values for type TListAssignOp

Value	Explanation
laAnd	Remove all elements not first second list
laCopy	Clear list and copy all strings from second list.
laDestUnique	Keep all elements that exists only in list2
laOr	Add all elements from second (and optional third) list, eliminate duplicates
laSrcUnique	Just keep all elements that exist only in source list
laXor	Remove elements in second lists, Add all elements from second list not in first list

This type determines what operation TList.Assign (336) or TFPList.assign (316) performs.

```
TListCallback = Types.TListCallback
```

`TListCallback` is the method callback prototype for the function that is passed to the `TFPList.ForEachCall` (317) 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 2.12: 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` (317) 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.

```
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 2.13: Enumeration values for type `TObjectTextEncoding`

Value	Explanation
<code>oteDFM</code>	Characters are in DFM (Delphi) format: widechar encoded.
<code>oteLFM</code>	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 2.14: 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 (350) class.

```
TPoint = Types.TPoint
```

This record describes a coordinate. It is used to handle the Top (290) and Left (290) properties of TComponent (290).

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 (364) 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 (350) descendent 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 (366) 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 2.15: Enumeration values for type TSeekOrigin

Value	Explanation
soBeginning	Offset is interpreted relative to the start of the stream.
soCurrent	Offset is interpreted relative to the current position in the stream.
soEnd	Offset is interpreted relative to the end of the stream.

Specifies the origin of the TStream.Seek (371) 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 (364) 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, ssExtra1, ssExtra2)
```

Table 2.16: 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
ssHyper	Hyper key pressed.
ssLeft	Left mouse button pressed.
ssMeta	Meta key pressed.
ssMiddle	Middle mouse button pressed.
ssNum	Num lock key pressed
ssQuad	Quadruple mouse click
ssRight	Right mouse button pressed.
ssScroll	Scroll lock key pressed
ssShift	Shift key pressed
ssSuper	Super key pressed.
ssTriple	Triple mouse click

Keyboard/Mouse shift state enumerator

```
TShortCut = (Word) .. (Word)
```

Enumeration type to identify shortcut key combinations.

```
TSmallPoint = record
  x : SmallInt;
  y : SmallInt;
end
```

Same as TPoint (206), only the X and Y ranges are limited to 2-byte integers instead of 4-byte integers.

```
TStreamOwnership = (soReference, soOwned)
```

Table 2.17: 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 (379) acts, when the adapter is freed.

TStreamProc = procedure(Stream: TStream) of object

Procedure type used in streaming.

```
TStringItem = record
    FString : string;
    FObject : TObject;
end
```

The TStringItem is used to store the string and object items in a TStringList (383) string list instance. It should never be used directly.

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.

TSynchronizeProcVar = procedure

Synchronize callback type

TThreadMethod = procedure of object

Procedure variable used when synchronizing threads.

```
TThreadPriority = (tpIdle, tpLowest, tpLower, tpNormal, tpHigher, tpHighest,
                  tpTimeCritical)
```

Table 2.18: 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.

```
TValueType = (vaNull, vaList, vaInt8, vaInt16, vaInt32, vaExtended, vaString,
             vaIdent, vaFalse, vaTrue, vaBinary, vaSet, vaLString, vaNil,
             vaCollection, vaSingle, vaCurrency, vaDate, vaWString, vaInt64,
             vaUTF8String, vaUString, vaQWord)
```

Table 2.19: Enumeration values for type TValueType

Value	Explanation
vaBinary	Binary data follows.
vaCollection	Collection follows
vaCurrency	Currency value follows
vaDate	Date value follows
vaExtended	Extended value.
vaFalse	Boolean False 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 True value.
vaUString	UnicodeString value
vaUTF8String	UTF8 encoded unicode string.
vaWString	Widestring value follows.

Enumerated type used to identify the kind of streamed property

```
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 (426) 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` (350) descendent 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.

2.3.3 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` (304) descendent 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` (304) 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` (304) descendent is freed.


```
WakeMainThread : TNotifyEvent = Nil
```

`WakeMainThread` is a handler, which, when set, is called by the `TThread.Synchronize` (409) 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` (213) 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.

2.4 Procedures and functions

2.4.1 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` (225), `GroupDescendentsWith` (217), `ClassGroupOf` (213)

2.4.2 BeginGlobalLoading

Synopsis: Not yet implemented

Declaration: `procedure BeginGlobalLoading`

Visibility: default

Description: Not yet implemented

2.4.3 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*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 \(217\)](#)

2.4.4 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` (207) 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 \(221\)](#)

2.4.5 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` (409) 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` (212) 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` (409), `WakeMainThread` (212)

2.4.6 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` (225), `ActivateClassGroup` (212), `GroupDescendentsWith` (217)

2.4.7 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

2.4.8 EndGlobalLoading

Synopsis: Not yet implemented.

Declaration: `procedure EndGlobalLoading`

Visibility: default

Description: Not yet implemented.

2.4.9 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.

2.4.10 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` (216) function does not raise an exception when it does not find the class, but returns a `Nil` pointer instead.

See also: `RegisterClass` (222), `GetClass` (216)

2.4.11 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` (202). 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` (202)

2.4.12 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` (217) is called.

Errors: `Nil` is returned if no handler is registered for the given type.

2.4.13 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` (218) is called.

Errors: `Nil` is returned if no handler is registered for the given type.

See also: `IntToIdent` (218), `TIntToIdent` (204), `FindIdentToInt` (215)

2.4.14 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` (216)

2.4.15 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` (214) function will raise an exception if it does not find the class.

See also: `RegisterClass` (222), `GetClass` (216)

2.4.16 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` (216), `GlobalFixupReferences` (216)

2.4.17 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` (216), `GlobalFixupReferences` (216)

2.4.18 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` (216), `GetFixupInstanceNames` (216)

2.4.19 GroupDescendentsWith

Synopsis: Add class to the group of another class.

Declaration: `procedure GroupDescendentsWith(AClass: TPersistentClass;
AClassGroup: TPersistentClass)`

Visibility: default

Description: `GroupDescendentsWith` adds `AClass` to the group that `AClassGroup` belongs to. If `AClassGroup` belongs to more than 1 group, then it is added to the group which contains the nearest ancestor.

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: `StartClassGroup` (225), `ActivateClassGroup` (212), `ClassGroupOf` (213)

2.4.20 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` (212)

2.4.21 IdentToInt

Synopsis: Looks up an integer value in a integer-to-identifier map list.

Declaration: `function IdentToInt(const Ident: string; var 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` (204), `TIntToIdent` (204), `IntToIdent` (218), `TIdentMapEntry` (204)

2.4.22 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` (220)

2.4.23 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 descendent. 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` (220), `ReadComponentResEx` (221), `ReadComponentResFile` (221)

2.4.24 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` (204), `TIntToIdent` (204), `IdentToInt` (217), `TIdentMapEntry` (204)

2.4.25 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` (206), `TSmallPoint` (208), `PointsEqual` (220)

2.4.26 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.

2.4.27 NotifyGlobalLoading

Synopsis: Not yet implemented.

Declaration: `procedure NotifyGlobalLoading`

Visibility: default

Description: Not yet implemented.

2.4.28 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` ([219](#))

2.4.29 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` ([219](#))

See also: `ObjectBinaryToText` ([219](#)), `ObjectTextToResource` ([219](#))

2.4.30 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.

2.4.31 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` (219), `ObjectResourceToText` (219)

2.4.32 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` (206) record with the given coordinates `AX` and `AY` filled in.

See also: `TPoint` (206), `SmallPoint` (225), `Rect` (221), `Bounds` (213)

2.4.33 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` (206) or `TSmallPoint` (208)) and returns `True` if the X and Y coordinates of the points are equal, or `False` otherwise.

See also: `TPoint` (206), `TSmallPoint` (208), `InvalidPoint` (218)

2.4.34 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` (221)

2.4.35 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` ([220](#))

2.4.36 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` ([373](#)) method to read the state of the component from the stream.

See also: `TStream.ReadComponentRes` ([373](#)), `WriteComponentResFile` ([226](#))

2.4.37 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` ([207](#)) 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` ([207](#)), `Point` ([220](#)), `SmallPoint` ([225](#)), `Bounds` ([213](#))

2.4.38 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` ([224](#))

2.4.39 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` ([222](#)), `RegisterClassAlias` ([222](#)), `RegisterComponents` ([222](#)), `UnregisterClass` ([225](#))

2.4.40 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` ([222](#)), `RegisterClasses` ([222](#)), `RegisterComponents` ([222](#)), `UnregisterClass` ([225](#))

2.4.41 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` ([222](#)), `RegisterClassAlias` ([222](#)), `RegisterComponents` ([222](#)), `UnregisterClass` ([225](#))

2.4.42 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` (222), `RegisterNoIcon` (224)

2.4.43 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` (215) is called, then this callback will be used to search for the component.

Errors: None.

See also: `FindGlobalComponent` (215), `UnRegisterFindGlobalComponentProc` (226)

2.4.44 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 descendents of `ComponentClass` in the `InitInheritedComponent` (218) call.

See also: `InitInheritedComponent` (218), `TInitComponentHandler` (204)

2.4.45 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` (217) and `IntToIdent` (218) can be used to implement these callback functions.

See also: `TIdentToInt` (204), `TIntToIdent` (204), `IdentToInt` (217), `IntToIdent` (218)

2.4.46 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` (222) except that it calls `RegisterNoIconProc` (211) instead of `RegisterComponentsProc` (211)

See also: `RegisterNoIconProc` (211), `RegisterComponents` (222)

2.4.47 RegisterNonActiveX

Synopsis: Register non-activex component.

Declaration: `procedure RegisterNonActiveX(ComponentClasses: Array of TComponentClass;
AxRegType: TActiveXRegType)`

Visibility: default

Description: Not yet implemented in Free Pascal

2.4.48 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` (221)

2.4.49 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.
`ences.a`

See also: `RedirectFixupReferences` (221), `RemoveFixupReferences` (224)

2.4.50 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` (208) record with the given coordinates `AX` and `AY` filled in.

See also: `TSmallPoint` (208), `Point` (220), `Rect` (221), `Bounds` (213)

2.4.51 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` (217), `ActivateClassGroup` (212), `ClassGroupOf` (213)

2.4.52 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` (225), `UnRegisterModuleClasses` (226), `RegisterClass` (222)

2.4.53 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.

2.4.54 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` (215) is called, then this callback will be no longer be used to search for the component.

Errors: None.

See also: `FindGlobalComponent` (215), `RegisterFindGlobalComponentProc` (223)

2.4.55 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` (225), `UnRegisterClasses` (225), `RegisterClasses` (222)

2.4.56 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` (373) method to write the state of the component to the stream.

See also: `TStream.WriteComponentRes` (373), `ReadComponentResFile` (221)

2.5 EBitsError

2.5.1 Description

When an index of a bit in a `TBits` (273) is out of the valid range (0 to `Count-1`) then a `EBitsError` exception is raised.

2.6 EClassNotFound

2.6.1 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` ([227](#))

2.7 EComponentError

2.7.1 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` ([290](#)), `TComponent.Name` ([300](#))

2.8 EFCreateError

2.8.1 Description

When the operating system reports an error during creation of a new file in the Filestream Constructor ([310](#)), a `EFCreateError` is raised.

See also: `EStreamError` ([229](#)), `EFOpenError` ([227](#))

2.9 EFileError

2.9.1 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` ([229](#)), `EReadError` ([229](#))

2.10 EFOpenError

2.10.1 Description

When the operating system reports an error during the opening of a file in the Filestream Constructor ([310](#)), a `EFOpenError` is raised.

See also: `EStreamError` ([229](#)), `EFCreateError` ([227](#))

2.11 EInvalidImage

2.11.1 Description

This exception is not used by Free Pascal but is provided for Delphi compatibility.

2.12 EInvalidOperation

2.12.1 Description

This exception is not used in Free Pascal, it is defined for Delphi compatibility purposes only.

2.13 EListError

2.13.1 Description

If an error occurs in one of the `TList` (330) or `TStrings` (388) 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.

See also: `TList` (330), `TStrings` (388)

2.14 EMethodNotFound

2.14.1 Description

This exception is no longer used in the streaming system. This error is replaced by a `EReadError` (229).

See also: `EFileError` (227), `EReadError` (229)

2.15 EObserver

2.15.1 Description

`EObserver` is an error that is raised when an object is registered as an observer, and it does not implement the `IFPObserver` (233) interface.

See also: `IFPObserver` (233), `IFPObserver.FPOAttachObserver` (232)

2.16 EOutOfResources

2.16.1 Description

This exception is not used in Free Pascal, it is defined for Delphi compatibility purposes only.

2.17 EParserError

2.17.1 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` ([343](#))

2.18 EReadError

2.18.1 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` ([227](#)), `EWriteError` ([230](#))

2.19 EResNotFound

2.19.1 Description

This exception is not used by Free Pascal but is provided for Delphi compatibility.

2.20 EStreamError

2.20.1 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` ([227](#))

2.21 EStringListError

2.21.1 Description

When an error occurs in one of the methods of TStrings (388) 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 (383) when TStringList.Duplicates (386) is False.

See also: TStrings (388), TStringList (383)

2.22 EThread

2.22.1 Description

Thread error exception.

2.23 EThreadDestroyCalled

2.23.1 Description

Exception raised when a thread is destroyed illegally.

2.24 EThreadExternalException

2.24.1 Description

EThreadExternalException is raised by for example TThread.CheckTerminated (410) and TThread.SetReturnValue (410) 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 (410), TThread.SetReturnValue (410)

2.25 EWriteError

2.25.1 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 (227), EReadError (229)

2.26 IDesignerNotify

2.26.1 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.

2.26.2 Method overview

Page	Property	Description
231	Modified	Notify that the component is modified.
231	Notification	Notification of owner changes

2.26.3 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.

2.26.4 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` ([292](#)) which is used to communicate adds to the components.

See also: `TComponent.Notification` ([292](#))

2.27 IFPObserved

2.27.1 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` ([195](#)) call, and must be detached using the `FPODetachObserver` ([195](#)) call.

This interface is not reference counted, so care must be taken that the `ooFree` message is sent with `FPONotifyObservers` ([195](#)) when the object is freed.

See also: `FPONotifyObservers` ([195](#))

2.27.2 Method overview

Page	Property	Description
232	FPOAttachObserver	Attach a new observer to the object
232	FPODetachObserver	Remove an observer from the list of observers.
232	FPONotifyObservers	Notify all observers

2.27.3 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` ([233](#)) interface. The `FPOObservedChanged` ([233](#)) method of the interface will be called whenever `FPONotifyObservers` ([195](#)) is used to notify observers of a change. Objects implementing this interface should check that `AObserver` actually implements the `IFPObserver` ([233](#)) 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` ([233](#)) interface, an `EObserver` ([228](#)) exception must be raised. No other errors should be raised, other than a possible out of memory error.

See also: `IFPObserver` ([233](#)), `FPOObservedChanged` ([233](#)), `FPONotifyObservers` ([195](#))

2.27.4 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` ([233](#)) is called.

If the object was added more than once using `FPOAttachObserver` ([195](#)), then it depends on the implementor of the interface whether or `FPODetachObserver` must be called an equal number of times.

See also: `IFPObserver` ([233](#)), `FPOObservedChanged` ([233](#)), `FPONotifyObservers` ([195](#)), `FPOAttachObserver` ([195](#))

2.27.5 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 `FPObservedChanged` (233) on the `IFPObserver` (233) 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` (233), `FPObservedChanged` (233), `FPODetachObserver` (195), `FPOAttachObserver` (195)

2.28 IFPObserver

2.28.1 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` (233). The change notifications arrive because the `FPObservedChanged` (195) method is called by the observed object.

See also: `IFPObserved` (231), `FPOAttachObserver` (232)

2.28.2 Method overview

Page	Property	Description
233	<code>FPObservedChanged</code>	Entry point for change notifications

2.28.3 IFPObserver.FPObservedChanged

Synopsis: Entry point for change notifications

Declaration: `procedure FPObservedChanged(ASender: TObject;
Operation: TFPObservedOperation;
Data: Pointer)`

Visibility: default

Description: `FPObservedChanged` is the method that is called by an observed object (`IFPObserved` (231)) when it calls `FPONotifyObservers` (232). The `Sender` is the object under observation, the `Operation` and `Data` are the parameters used in the call to `FPONotifyObservers`.

See also: `IFPObserved` (231), `FPONotifyObservers` (232)

2.29 IInterfaceComponentReference

2.29.1 Description

`IInterfaceComponentReference` is an interface to return the component that implements a given interface. It is implemented by `TComponent` (290).

See also: `TComponent` (290)

2.29.2 Method overview

Page	Property	Description
234	GetComponent	Return component instance

2.29.3 IInterfaceComponentReference.GetComponent

Synopsis: Return component instance

Declaration: `function GetComponent : TComponent`

Visibility: default

Description: `GetComponent` returns the component instance.

Errors: None.

See also: `TComponent` ([290](#))

2.30 IInterfaceList

2.30.1 Description

`IInterfaceList` is an interface for maintaining a list of interfaces, strongly resembling the standard `TList` ([330](#)) 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` ([330](#))

2.30.2 Method overview

Page	Property	Description
237	Add	Add an interface to the list
236	Clear	Clear the list
236	Delete	Remove an interface from the list
237	Exchange	Exchange 2 interfaces in the list
237	First	Return the first non-empty interface in the list.
235	Get	Retrieve an interface pointer from the list.
235	GetCapacity	Return the capacity of the list.
235	GetCount	Return the current number of elements in the list.
237	IndexOf	Return the index of an interface.
237	Insert	Insert an interface in the list.
238	Last	Returns the last non-nil interface in the list.
238	Lock	Lock the list
235	Put	Write an item to the list
238	Remove	Remove an interface from the list
236	SetCapacity	Set the capacity of the list
236	SetCount	Set the number of items in the list
238	Unlock	Unlock the list.

2.30.3 Property overview

Page	Property	Access	Description
238	Capacity	rw	Capacity of the list
239	Count	rw	Current number of elements in the list.
239	Items	rw	Provides Index-based, sequential, access to the interfaces in the list.

2.30.4 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` ([239](#)) property.

See also: `IInterfaceList.Items` ([239](#)), `TList.Items` ([337](#))

2.30.5 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` ([238](#)) property.

See also: `IInterfaceList.Capacity` ([238](#)), `TList.Capacity` ([337](#))

2.30.6 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` ([239](#)) property.

See also: `IInterfaceList.Count` ([239](#)), `TList.Count` ([337](#))

2.30.7 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` ([239](#)) property.

See also: `IInterfaceList.Items` ([239](#)), `TList.Items` ([337](#))

2.30.8 **IInterfaceList.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` (238) property.

See also: `IInterfaceList.Capacity` (238), `TList.Capacity` (337)

2.30.9 **IInterfaceList.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` (238)

See also: `IInterfaceList.Count` (239), `TList.Count` (337)

2.30.10 **IInterfaceList.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` (333)

2.30.11 **IInterfaceList.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` (333), `IInterfaceList.Add` (237), `IInterfaceList.Delete` (236), `IInterfaceList.Insert` (237)

2.30.12 **IInterfaceList.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` (333), `IInterfaceList.Add` (237), `IInterfaceList.Delete` (236), `IInterfaceList.Insert` (237)

2.30.13 **IInterfaceList.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` (334), `IInterfaceList.IndexOf` (237), `IInterfaceList.Last` (238)

2.30.14 **IInterfaceList.IndexOf**

Synopsis: Return the index of an interface.

Declaration: `function IndexOf(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` (335), `IInterfaceList.First` (237), `IInterfaceList.Last` (238)

2.30.15 **IInterfaceList.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` (332), `IInterfaceList.Insert` (237), `IInterfaceList.Delete` (236)

2.30.16 **IInterfaceList.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` (335), `IInterfaceList.Add` (237), `IInterfaceList.Delete` (236)

2.30.17 **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` ([335](#)), `IInterfaceList.First` ([237](#)), `IInterfaceList.IndexOf` ([237](#))

2.30.18 **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` ([336](#)), `IInterfaceList.Delete` ([236](#)), `IInterfaceList.IndexOf` ([237](#))

2.30.19 **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` ([238](#)) is called.

See also: `TInterfaceList.Lock` ([327](#)), `IInterfaceList.Unlock` ([238](#))

2.30.20 **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` ([327](#)), `IInterfaceList.Lock` ([238](#))

2.30.21 **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` ([337](#)), `IInterfaceList.Count` ([239](#))

2.30.22 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` ([238](#)), `TList.Count` ([337](#))

2.30.23 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` ([239](#)), `TList.Items` ([337](#))

2.31 IStreamPersist

2.31.1 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` ([350](#)), `TComponent` ([290](#)), `TStream` ([369](#))

2.31.2 Method overview

Page	Property	Description
240	<code>LoadFromStream</code>	Load persistent data from stream.
240	<code>SaveToStream</code>	Save persistent data to stream.

2.31.3 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 \(350\)](#), [TComponent \(290\)](#), [TStream \(369\)](#), [IStreamPersist.SaveToStream \(240\)](#)

2.31.4 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 \(350\)](#), [TComponent \(290\)](#), [TStream \(369\)](#), [IStreamPersist.LoadFromStream \(240\)](#)

2.32 IStringsAdapter

2.32.1 Description

Is not yet supported in Free Pascal.

See also: [TStrings \(388\)](#)

2.32.2 Method overview

Page	Property	Description
240	<code>ReferenceStrings</code>	Add a reference to the indicated strings.
240	<code>ReleaseStrings</code>	Release the reference to the strings.

2.32.3 IStringsAdapter.ReferenceStrings

Synopsis: Add a reference to the indicated strings.

Declaration: `procedure ReferenceStrings(S: TStrings)`

Visibility: default

2.32.4 IStringsAdapter.ReleaseStrings

Synopsis: Release the reference to the strings.

Declaration: `procedure ReleaseStrings`

Visibility: default

2.33 IVCLComObject

2.33.1 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 ([300](#))

2.33.2 Method overview

Page	Property	Description
242	FreeOnRelease	Is called by TComponent.FreeOnRelease
242	GetIDsOfNames	The IDispatch: GetIDsOfNames call for automation servers.
241	GetTypeInfo	The IDispatch: GetTypeInfo call for automation servers
241	GetTypeInfoCount	The IDispatch: GetTypeInfoCount call for automation servers
242	Invoke	The IDispatch: Invoke call for automation servers.
242	SafeCallException	This method can be invoked if an exception occurs during Invoke

2.33.3 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 ([241](#))

2.33.4 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 ([241](#))

2.33.5 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` ([242](#))

2.33.6 IVCLComObject.Invoke

Synopsis: The `IDispatch::Invoke` call for automation servers.

Declaration: `function Invoke(DispID: Integer; const IID: TGuid; LocaleID: Integer;
Flags: Word; var Params; var Result: 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` ([242](#))

2.33.7 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` ([242](#))

2.33.8 IVCLComObject.FreeOnRelease

Synopsis: Is called by `TComponent.FreeOnRelease`

Declaration: `procedure FreeOnRelease`

Visibility: default

Description: `FreeOnRelease` is called by `TComponent.FreeOnRelease` (295) for the `IVCLComObject` interface implemented by `TComponent`.

See also: `TComponent.FreeOnRelease` (295)

2.34 TAbstractObjectReader

2.34.1 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 descendent classes.

Different streaming mechanisms can be implemented by making a descendent from `TAbstractObjectReader`. The `TBinaryObjectReader` (261) class is such a descendent class, which streams data in binary (Delphi compatible) format.

All methods described in this class, mustbe implemented by descendent classes.

See also: `TBinaryObjectReader` (261)

2.34.2 Method overview

Page	Property	Description
244	<code>BeginComponent</code>	Marks the reading of a new component.
245	<code>BeginProperty</code>	Marks the reading of a property value.
244	<code>BeginRootComponent</code>	Starts the reading of the root component.
243	<code>NextValue</code>	Returns the type of the next value in the stream.
245	<code>Read</code>	Read raw data from stream
245	<code>ReadBinary</code>	Read binary data from the stream.
246	<code>ReadCurrency</code>	Read a currency value from the stream.
246	<code>ReadDate</code>	Read a date value from the stream.
245	<code>ReadFloat</code>	Read a float value from the stream.
247	<code>ReadIdent</code>	Read an identifier from the stream.
247	<code>ReadInt16</code>	Read a 16-bit integer from the stream.
248	<code>ReadInt32</code>	Read a 32-bit integer from the stream.
248	<code>ReadInt64</code>	Read a 64-bit integer from the stream.
247	<code>ReadInt8</code>	Read an 8-bit integer from the stream.
248	<code>ReadSet</code>	Reads a set from the stream.
246	<code>ReadSingle</code>	Read a single (real-type) value from the stream.
249	<code>ReadStr</code>	Read a shortstring from the stream
249	<code>ReadString</code>	Read a string of type <code>StringType</code> from the stream.
250	<code>ReadUnicodeString</code>	Read a unicode string value
244	<code>ReadValue</code>	Reads the type of the next value.
249	<code>ReadWideString</code>	Read a widestring value from the stream.
250	<code>SkipComponent</code>	Skip till the end of the component.
250	<code>SkipValue</code>	Skip the current value.

2.34.3 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 ([244](#))

2.34.4 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 ([244](#))

2.34.5 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 ([244](#)).

See also: TAbstractObjectReader.BeginComponent ([244](#))

2.34.6 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.

Descendent 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 descendent forms.

See also: TAbstractObjectReader.BeginRootComponent ([244](#)), TAbstractObjectReader.BeginProperty ([245](#))

2.34.7 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` (244)

2.34.8 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` (264), `TReader.Read` (358)

2.34.9 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` (244) returned a valuetype of `vaBinary`). The data should be stored in the `DestData` memory stream by descendent classes.

See also: `TAbstractObjectReader.ReadFloat` (245), `TAbstractObjectReader.ReadDate` (246), `TAbstractObjectReader.ReadSingle` (246), `TAbstractObjectReader.ReadIdent` (247), `TAbstractObjectReader.ReadInt8` (247), `TAbstractObjectReader.ReadInt16` (247), `TAbstractObjectReader.ReadInt32` (248), `TAbstractObjectReader.ReadInt64` (248), `TabstractObjectReader.ReadSet` (248), `TabstractObjectReader.ReadStr` (249), `TabstractObjectReader.ReadString` (249)

2.34.10 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` (244) returned a valuetype of `vaExtended`). The return value should be the value of the float.

See also: `TAbstractObjectReader.ReadFloat` (245), `TAbstractObjectReader.ReadDate` (246), `TAbstractObjectReader.ReadSingle` (246), `TAbstractObjectReader.ReadIdent` (247), `TAbstractObjectReader.ReadInt8` (247), `TAbstractObjectReader.ReadInt16` (247), `TAbstractObjectReader.ReadInt32` (248), `TAbstractObjectReader.ReadInt64` (248), `TAbstractObjectReader.ReadSet` (248), `TAbstractObjectReader.ReadStr` (249), `TAbstractObjectReader.ReadString` (249)

2.34.11 `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` (244) returned a valuetype of `vaSingle`). The return value should be the value of the float.

See also: `TAbstractObjectReader.ReadFloat` (245), `TAbstractObjectReader.ReadDate` (246), `TAbstractObjectReader.ReadSingle` (246), `TAbstractObjectReader.ReadIdent` (247), `TAbstractObjectReader.ReadInt8` (247), `TAbstractObjectReader.ReadInt16` (247), `TAbstractObjectReader.ReadInt32` (248), `TAbstractObjectReader.ReadInt64` (248), `TAbstractObjectReader.ReadSet` (248), `TAbstractObjectReader.ReadStr` (249), `TAbstractObjectReader.ReadString` (249)

2.34.12 `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` (244) returned a valuetype 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` (245), `TAbstractObjectReader.ReadSingle` (246), `TAbstractObjectReader.ReadIdent` (247), `TAbstractObjectReader.ReadInt8` (247), `TAbstractObjectReader.ReadInt16` (247), `TAbstractObjectReader.ReadInt32` (248), `TAbstractObjectReader.ReadInt64` (248), `TAbstractObjectReader.ReadSet` (248), `TAbstractObjectReader.ReadStr` (249), `TAbstractObjectReader.ReadString` (249)

2.34.13 `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 descendent classes, and should return the currency value read from the stream.

See also: `TAbstractObjectWriter.WriteCurrency` (253)

2.34.14 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` (244) 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 2.20:

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` (245), `TAbstractObjectReader.ReadDate` (246), `TAbstractObjectReader.ReadSingle` (246), `TAbstractObjectReader.ReadInt8` (247), `TAbstractObjectReader.ReadInt16` (247), `TAbstractObjectReader.ReadInt32` (248), `TAbstractObjectReader.ReadInt64` (248), `TabstractObjectReader.ReadSet` (248), `TabstractObjectReader.ReadStr` (249), `TabstractObjectReader.ReadString` (249)

2.34.15 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` (244) 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` (245), `TAbstractObjectReader.ReadDate` (246), `TAbstractObjectReader.ReadSingle` (246), `TAbstractObjectReader.ReadIdent` (247), `TAbstractObjectReader.ReadInt16` (247), `TAbstractObjectReader.ReadInt32` (248), `TAbstractObjectReader.ReadInt64` (248), `TabstractObjectReader.ReadSet` (248), `TabstractObjectReader.ReadStr` (249), `TabstractObjectReader.ReadString` (249)

2.34.16 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` (244) 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` (245), `TAbstractObjectReader.ReadDate` (246), `TAbstractObjectReader.ReadSingle` (246), `TAbstractObjectReader.ReadIdent` (247), `TAbstractObjectReader.ReadInt8` (247), `TAbstractObjectReader.ReadInt32` (248), `TAbstractObjectReader.ReadInt64` (248), `TabstractObjectReader.ReadSet` (248), `TabstractObjectReader.ReadStr` (249), `TabstractObjectReader.ReadString` (249)

2.34.17 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` (244) 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` (245), `TAbstractObjectReader.ReadDate` (246), `TAbstractObjectReader.ReadSingle` (246), `TAbstractObjectReader.ReadIdent` (247), `TAbstractObjectReader.ReadInt8` (247), `TAbstractObjectReader.ReadInt16` (247), `TAbstractObjectReader.ReadInt32` (248), `TabstractObjectReader.ReadSet` (248), `TabstractObjectReader.ReadStr` (249), `TabstractObjectReader.ReadString` (249)

2.34.18 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` (244) 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` (245), `TAbstractObjectReader.ReadDate` (246), `TAbstractObjectReader.ReadSingle` (246), `TAbstractObjectReader.ReadIdent` (247), `TAbstractObjectReader.ReadInt8` (247), `TAbstractObjectReader.ReadInt16` (247), `TAbstractObjectReader.ReadInt32` (248), `TabstractObjectReader.ReadSet` (248), `TabstractObjectReader.ReadStr` (249), `TabstractObjectReader.ReadString` (249)

2.34.19 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` (244) 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` (245), `TAbstractObjectReader.ReadDate` (246), `TAbstractObjectReader.ReadSingle` (246), `TAbstractObjectReader.ReadIdent` (247), `TAbstractObjectReader.ReadInt8` (247), `TAbstractObjectReader.ReadInt16` (247), `TAbstractObjectReader.ReadInt32` (248), `TAbstractObjectReader.ReadInt64` (248), `TabstractObjectReader.ReadStr` (249), `TabstractObjectReader.ReadString` (249)

2.34.20 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` (244) returned a valuetype of `vaLString`, `vaWString` or `vaString`). The return value is the string.

See also: `TAbstractObjectReader.ReadFloat` (245), `TAbstractObjectReader.ReadDate` (246), `TAbstractObjectReader.ReadSingle` (246), `TAbstractObjectReader.ReadIdent` (247), `TAbstractObjectReader.ReadInt8` (247), `TAbstractObjectReader.ReadInt16` (247), `TAbstractObjectReader.ReadInt32` (248), `TAbstractObjectReader.ReadInt64` (248), `TabstractObjectReader.ReadSet` (248), `TabstractObjectReader.ReadString` (249)

2.34.21 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` (244) returned a valuetype of `vaLString`, `vaWString` or `vaString`). The return value is the string.

See also: `TAbstractObjectReader.ReadFloat` (245), `TAbstractObjectReader.ReadDate` (246), `TAbstractObjectReader.ReadSingle` (246), `TAbstractObjectReader.ReadIdent` (247), `TAbstractObjectReader.ReadInt8` (247), `TAbstractObjectReader.ReadInt16` (247), `TAbstractObjectReader.ReadInt32` (248), `TAbstractObjectReader.ReadInt64` (248), `TabstractObjectReader.ReadSet` (248), `TabstractObjectReader.ReadStr` (249)

2.34.22 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 descendent classes.

See also: `TAbstractObjectWriter.WriteString` ([255](#))

2.34.23 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).

Descendent classes should override this method to actually read a `UnicodeString` value.

See also: `TBinaryObjectWriter.WriteUnicodeString` ([272](#)), `TAbstractObjectReader.ReadWideString` ([249](#))

2.34.24 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 descendent of `TAbstractObjectReader` should implement this in a way which is optimal for the implemented stream format.

See also: `TAbstractObjectReader.BeginComponent` ([244](#)), `TAbstractObjectReader.SkipValue` ([250](#))

2.34.25 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` ([250](#))

2.35 TAbstractObjectWriter

2.35.1 Description

Abstract driver class for writing component data.

2.35.2 Method overview

Page	Property	Description
251	BeginCollection	Start writing a collection.
251	BeginComponent	Start writing a component
251	BeginList	Start writing a list.
252	BeginProperty	Start writing a property
252	EndList	Mark the end of a list.
252	EndProperty	Marks the end of writing of a property.
252	Write	Write raw data to stream
252	WriteBinary	Writes binary data to the stream.
253	WriteBoolean	Writes a boolean value to the stream.
253	WriteCurrency	Write a currency value to the stream
253	WriteDate	Writes a date type to the stream.
253	WriteFloat	Writes a float value to the stream.
254	WriteIdent	Writes an identifier to the stream.
254	WriteInteger	Writes an integer value to the stream
254	WriteMethodName	Writes a methodname to the stream.
255	WriteSet	Writes a set value to the stream.
253	WriteSingle	Writes a single-type real value to the stream.
255	WriteString	Writes a string value to the stream.
254	WriteUInt64	Write an unsigned 64-bit integer
255	WriteUnicodeString	Write a unicode string to the stream.
254	WriteVariant	Write a variant to the stream
255	WriteWideString	Write a widestring value to the stream

2.35.3 TAbstractObjectWriter.BeginCollection

Synopsis: Start writing a collection.

Declaration: `procedure BeginCollection; Virtual; Abstract`

Visibility: `public`

Description: Start writing a collection.

2.35.4 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

2.35.5 TAbstractObjectWriter.BeginList

Synopsis: Start writing a list.

Declaration: `procedure BeginList; Virtual; Abstract`

Visibility: `public`

Description: Start writing a list.

2.35.6 TAbstractObjectWriter.EndList

Synopsis: Mark the end of a list.

Declaration: `procedure EndList; Virtual; Abstract`

Visibility: `public`

Description: Mark the end of a list.

2.35.7 TAbstractObjectWriter.BeginProperty

Synopsis: Start writing a property

Declaration: `procedure BeginProperty(const PropName: string); Virtual; Abstract`

Visibility: `public`

Description: Start writing a property

2.35.8 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.

2.35.9 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` ([270](#)), `TWriter.Write` ([421](#))

2.35.10 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.

2.35.11 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.

2.35.12 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.

2.35.13 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.

2.35.14 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.

2.35.15 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 descendent classes.

See also: `TAbstractObjectReader.ReadCurrency` ([246](#))

2.35.16 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.

2.35.17 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

2.35.18 TAbstractObjectWriter.WriteUInt64

Synopsis: Write an unsigned 64-bit integer

Declaration: `procedure WriteUInt64(Value: QWord); Virtual; Abstract`

Visibility: public

Description: `WriteUInt64` must be overridden by descendent classes to write a 64-bit unsigned Value (value type `QWord`) to the stream.

Errors: None.

See also: `TBinaryObjectWriter.WriteUInt64` ([271](#))

2.35.19 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 descendent classes to write a simple variant type to the stream. `WriteVariant` does not write arrays types or complex types.

See also: `TBinaryObjectWriter.WriteVariant` ([272](#))

2.35.20 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.

2.35.21 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.

2.35.22 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.

2.35.23 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 descendent classes.

See also: `TAbstractObjectReader.ReadWideString` ([249](#))

2.35.24 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 descendent classes to write a unicodestring (value type `vaUString`) value to the stream.

See also: `TBinaryObjectWriter.WriteUnicodeString` ([272](#))

2.36 TBasicAction

2.36.1 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 descendent class and create an instance of this class instead.

See also: `TBasicActionLink` ([259](#)), `TComponent` ([290](#))

2.36.2 Method overview

Page	Property	Description
256	Create	Creates a new instance of a TBasicAction (255) class.
256	Destroy	Destroys the action.
257	Execute	Triggers the OnExecute (259) event
257	ExecuteTarget	Executes the action on the Target object
256	HandlesTarget	Determines whether Target can be handled by this action
258	RegisterChanges	Registers a new client with the action.
258	UnRegisterChanges	Unregisters a client from the list of clients
258	Update	Triggers the OnUpdate (259) event
257	UpdateTarget	Notify client controls when the action updates itself.

2.36.3 Property overview

Page	Property	Access	Description
258	ActionComponent	rw	Returns the component that initiated the action.
259	OnExecute	rw	Event triggered when the action executes.
259	OnUpdate	rw	Event triggered when the application is idle.

2.36.4 TBasicAction.Create

Synopsis: Creates a new instance of a TBasicAction ([255](#)) 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 descendent manually, actions are created in an IDE.

See also: `Destroy` ([256](#)), `AssignClient` ([255](#))

2.36.5 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.

2.36.6 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; descendent classes should override this method to implement appropriate checks.

See also: `UpdateTarget` ([257](#)), `ExecuteTarget` ([257](#))

2.36.7 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. Descendent 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` ([256](#)), `ExecuteTarget` ([257](#))

2.36.8 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. Descendent 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` ([256](#)), `UpdateTarget` ([257](#)), `Execute` ([257](#))

2.36.9 TBasicAction.Execute

Synopsis: Triggers the `OnExecute` ([259](#)) 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.

2.36.10 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` ([258](#))

2.36.11 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` ([258](#)), `Destroy` ([256](#))

2.36.12 TBasicAction.Update

Synopsis: Triggers the `OnUpdate` ([259](#)) 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` ([259](#)) event, and perform any checks in that handler.

See also: `OnUpdate` ([259](#)), `Execute` ([257](#)), `UpdateTarget` ([257](#))

2.36.13 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 toolbutton 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` ([257](#)), `OnExecute` ([259](#))

2.36.14 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 (257) 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 (257), OnUpdate (259)

2.36.15 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 (258), OnExecute (259)

2.37 TBasicActionLink

2.37.1 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 descendents of TBasicActionLink which communicate changes in the action to the component.

See also: TBasicAction (255)

2.37.2 Method overview

Page	Property	Description
260	Create	Creates a new instance of the TBasicActionLink class
260	Destroy	Destroys the TBasicActionLink instance.
260	Execute	Calls the action's Execute method.
261	Update	Calls the action's Update method

2.37.3 Property overview

Page	Property	Access	Description
261	Action	rw	The action to which the link was assigned.
261	OnChange	rw	Event handler triggered when the action's properties change

2.37.4 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` ([260](#))

2.37.5 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` ([260](#))

2.37.6 TBasicActionLink.Execute

Synopsis: Calls the action's `Execute` method.

Declaration: `function Execute(AComponent: TComponent) : Boolean; Virtual`

Visibility: public

Description: `Execute` sets the `ActionComponent` ([258](#)) property of the associated Action ([261](#)) to `AComponent` and then calls the Action's `execute` ([257](#)) 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 ([261](#)), TBasicAction.ActionComponent ([258](#)), TBasicAction.Execute ([257](#)), TBasicAction.onExecute ([259](#))

2.37.7 TBasicActionLink.Update

Synopsis: Calls the action's Update method

Declaration: `function Update : Boolean; Virtual`

Visibility: `public`

Description: `Update` calls the associated Action's `Update` (258) method.

Component programmers can override the `Update` method to provide additional processing when the `Update` method occurs.

2.37.8 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 (255) 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` (255)

2.37.9 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` (259), `TBasicAction.Change` (255)

2.38 TBinaryObjectReader

2.38.1 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` (243). 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` (373) call should be used.

See also: `TAbstractObjectReader` (243), `TBinaryObjectWriter` (268)

2.38.2 Method overview

Page	Property	Description
263	<code>BeginComponent</code>	Start reading a component.
263	<code>BeginProperty</code>	Start reading a property.
263	<code>BeginRootComponent</code>	Start reading the root component.
262	<code>Create</code>	Creates a new binary data reader instance.
262	<code>Destroy</code>	Destroys the binary data reader.
263	<code>NextValue</code>	Return the type of the next value.
264	<code>Read</code>	Read raw data from stream
264	<code>ReadBinary</code>	Start reading a binary value.
265	<code>ReadCurrency</code>	Read a currency value from the stream.
264	<code>ReadDate</code>	Read a date.
264	<code>ReadFloat</code>	Read a float value
265	<code>ReadIdent</code>	Read an identifier
265	<code>ReadInt16</code>	Read a 16-bits integer.
266	<code>ReadInt32</code>	Read a 32-bits integer.
266	<code>ReadInt64</code>	Read a 64-bits integer.
265	<code>ReadInt8</code>	Read an 8-bits integer.
266	<code>ReadSet</code>	Read a set
264	<code>ReadSingle</code>	Read a single-size float value
266	<code>ReadStr</code>	Read a short string
266	<code>ReadString</code>	Read a string
267	<code>ReadUnicodeString</code>	Read a unicode string value
263	<code>ReadValue</code>	Read the next value in the stream
267	<code>ReadWideString</code>	Read a widestring value from the stream.
267	<code>SkipComponent</code>	Skip a component's data
267	<code>SkipValue</code>	Skip a value's data

2.38.3 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` ([243](#))

2.38.4 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` ([262](#))

2.38.5 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` (243)

2.38.6 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` (243)

2.38.7 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` (243)

2.38.8 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` (243)

2.38.9 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` (243)

2.38.10 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` ([245](#)), `TReader.Read` ([358](#))

2.38.11 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` ([243](#))

2.38.12 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` ([243](#))

2.38.13 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` ([243](#))

2.38.14 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` (243)

2.38.15 `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` (243).

See also: `TAbstractObjectReader.ReadCurrency` (246), `TBinaryObjectWriter.WriteCurrency` (271)

2.38.16 `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` (243)

2.38.17 `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` (243)

2.38.18 `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` (243)

2.38.19 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` ([243](#))

2.38.20 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` ([243](#))

2.38.21 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` ([243](#))

2.38.22 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` ([243](#))

2.38.23 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` (243)

2.38.24 `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` (243).

See also: `TAbstractObjectReader.ReadWideString` (249), `TBinaryObjectWriter.WriteWideString` (272)

2.38.25 `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` (250)

2.38.26 `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` (243).

2.38.27 `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` (243)

2.39 TBinaryObjectWriter

2.39.1 Description

Driver class which stores component data in binary form.

2.39.2 Method overview

Page	Property	Description
269	BeginCollection	Start writing a collection.
269	BeginComponent	Start writing a component
269	BeginList	Start writing a list.
269	BeginProperty	Start writing a property
268	Create	Creates a new instance of a binary object writer.
268	Destroy	Destroys an instance of the binary object writer.
269	EndList	Mark the end of a list.
269	EndProperty	Marks the end of writing of a property.
270	Write	Write raw data to stream
270	WriteBinary	Writes binary data to the stream.
270	WriteBoolean	Writes a boolean value to the stream.
271	WriteCurrency	Write a currency-valued type to a stream
270	WriteDate	Writes a date type to the stream.
270	WriteFloat	Writes a float value to the stream.
271	WriteIdent	Writes an identifier to the stream.
271	WriteInteger	Writes an integer value to the stream.
271	WriteMethodName	Writes a methodname to the stream.
271	WriteSet	Writes a set value to the stream.
270	WriteSingle	Writes a single-type real value to the stream.
272	WriteStr	Write a string to the binary stream
272	WriteString	Writes a string value to the stream.
271	WriteUInt64	Write an unsigned 64-bit integer
272	WriteUnicodeString	Write a unicode string to the stream.
272	WriteVariant	Write a variant to the stream
272	WriteWideString	Write a widestring-valued type to a stream

2.39.3 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.

2.39.4 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.

2.39.5 TBinaryObjectWriter.BeginCollection

Synopsis: Start writing a collection.

Declaration: `procedure BeginCollection; Override`

Visibility: `public`

2.39.6 TBinaryObjectWriter.BeginComponent

Synopsis: Start writing a component

Declaration: `procedure BeginComponent(Component: TComponent; Flags: TFileFlags;
ChildPos: Integer); Override`

Visibility: `public`

2.39.7 TBinaryObjectWriter.BeginList

Synopsis: Start writing a list.

Declaration: `procedure BeginList; Override`

Visibility: `public`

2.39.8 TBinaryObjectWriter.EndList

Synopsis: Mark the end of a list.

Declaration: `procedure EndList; Override`

Visibility: `public`

2.39.9 TBinaryObjectWriter.BeginProperty

Synopsis: Start writing a property

Declaration: `procedure BeginProperty(const PropName: string); Override`

Visibility: `public`

2.39.10 TBinaryObjectWriter.EndProperty

Synopsis: Marks the end of writing of a property.

Declaration: `procedure EndProperty; Override`

Visibility: `public`

2.39.11 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` ([252](#)), `TWriter.Write` ([421](#))

2.39.12 TBinaryObjectWriter.WriteBinary

Synopsis: Writes binary data to the stream.

Declaration: `procedure WriteBinary(const Buffer; Count: LongInt); Override`

Visibility: public

2.39.13 TBinaryObjectWriter.WriteBoolean

Synopsis: Writes a boolean value to the stream.

Declaration: `procedure WriteBoolean(Value: Boolean); Override`

Visibility: public

2.39.14 TBinaryObjectWriter.WriteFloat

Synopsis: Writes a float value to the stream.

Declaration: `procedure WriteFloat(const Value: Extended); Override`

Visibility: public

2.39.15 TBinaryObjectWriter.WriteSingle

Synopsis: Writes a single-type real value to the stream.

Declaration: `procedure WriteSingle(const Value: Single); Override`

Visibility: public

2.39.16 TBinaryObjectWriter.WriteDate

Synopsis: Writes a date type to the stream.

Declaration: `procedure WriteDate(const Value: TDateTime); Override`

Visibility: public

2.39.17 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` (250).

See also: `TAbstractObjectWriter.WriteCurrency` (253)

2.39.18 TBinaryObjectWriter.WriteIdent

Synopsis: Writes an identifier to the stream.

Declaration: `procedure WriteIdent(const Ident: string); Override`

Visibility: public

2.39.19 TBinaryObjectWriter.WriteInteger

Synopsis: Writes an integer value to the stream.

Declaration: `procedure WriteInteger(Value: Int64); Override`

Visibility: public

2.39.20 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` (254)

2.39.21 TBinaryObjectWriter.WriteMethodName

Synopsis: Writes a methodname to the stream.

Declaration: `procedure WriteMethodName(const Name: string); Override`

Visibility: public

2.39.22 TBinaryObjectWriter.WriteSet

Synopsis: Writes a set value to the stream.

Declaration: `procedure WriteSet(Value: LongInt; SetType: Pointer); Override`

Visibility: public

2.39.23 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`.

2.39.24 TBinaryObjectWriter.WriteString

Synopsis: Writes a string value to the stream.

Declaration: `procedure WriteString(const Value: string); Override`

Visibility: public

2.39.25 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` (250).

See also: `TAbstractObjectWriter.WriteWidestring` (255)

2.39.26 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 (value type `vaUString`) value to the stream. It simply writes the character length and then all widecharacters.

See also: `TAbstractObjectWriter.WriteUnicodeString` (255)

2.39.27 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.

2.40 TBits

2.40.1 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 than 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.

2.40.2 Method overview

Page	Property	Description
275	<code>AndBits</code>	Performs an <code>and</code> operation on the bits.
274	<code>Clear</code>	Clears a particular bit.
275	<code>Clearall</code>	Clears all bits in the array.
273	<code>Create</code>	Creates a new bits collection.
274	<code>Destroy</code>	Destroys a bit collection
277	<code>Equals</code>	Determines whether the bits of 2 arrays are equal.
277	<code>FindFirstBit</code>	Find first bit with a particular value
278	<code>FindNextBit</code>	Searches the next bit with a particular value.
278	<code>FindPrevBit</code>	Searches the previous bit with a particular value.
276	<code>Get</code>	Retrieve the value of a particular bit
274	<code>GetFSIZE</code>	Returns the number of records used to store the bits.
276	<code>Grow</code>	Expands the bits array to the requested size.
276	<code>NotBits</code>	Performs a <code>not</code> operation on the bits.
278	<code>OpenBit</code>	Returns the position of the first bit that is set to <code>False</code> .
275	<code>OrBits</code>	Performs an <code>or</code> operation on the bits.
277	<code>SetIndex</code>	Sets the start position for <code>FindNextBit</code> (278) and <code>FindPrevBit</code> (278)
274	<code>SetOn</code>	Turn a particular bit on.
275	<code>XorBits</code>	Performs a <code>xor</code> operation on the bits.

2.40.3 Property overview

Page	Property	Access	Description
279	<code>Bits</code>	<code>rw</code>	Access to all bits in the array.
279	<code>Size</code>	<code>rw</code>	Current size of the array of bits.

2.40.4 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` ([274](#))

2.40.5 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` (273)

2.40.6 TBits.GetFSize

Synopsis: Returns the number of records used to store the bits.

Declaration: `function GetFSize : LongInt`

Visibility: `public`

Description: `GetFSize` returns the number of records used to store the current number of bits.

Errors: None.

See also: `Size` (279)

2.40.7 TBits.SetOn

Synopsis: Turn a particular bit on.

Declaration: `procedure SetOn(Bit: LongInt)`

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` (276).

Errors: If `bit` is larger than the maximum allowed bits array size or is negative, an `EBitsError` (226) exception is raised.

See also: `Bits` (279), `Clear` (274)

2.40.8 TBits.Clear

Synopsis: Clears a particular bit.

Declaration: `procedure Clear(Bit: LongInt)`

Visibility: `public`

Description: `Clear` clears the bit at position `bit`. If the array If `bit` is at a position bigger than the current size, the collection is expanded to the required size using `Grow` (276).

Errors: If `bit` is larger than the maximum allowed bits array size or is negative, an `EBitsError` (226) exception is raised.

See also: `Bits` (279), `seton` (274)

2.40.9 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 \(279\)](#), [clear \(274\)](#)

2.40.10 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 \(275\)](#), [OrBits \(275\)](#), [XOrBits \(275\)](#), [NotBits \(276\)](#)

2.40.11 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 \(275\)](#), [andBits \(275\)](#), [XOrBits \(275\)](#), [NotBits \(276\)](#)

2.40.12 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` ([275](#)), `andBits` ([275](#)), `OrBits` ([275](#)), `NotBits` ([276](#))

2.40.13 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` ([275](#)), `andBits` ([275](#)), `OrBits` ([275](#)), `XOrBits` ([275](#))

2.40.14 TBits.Get

Synopsis: Retrieve the value of a particular bit

Declaration: `function Get(Bit: LongInt) : 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` ([226](#)) exception is raised.

See also: `Bits` ([279](#)), `FindFirstBit` ([277](#)), `seton` ([274](#))

2.40.15 TBits.Grow

Synopsis: Expands the bits array to the requested size.

Declaration: `procedure Grow(NBit: LongInt)`

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` ([226](#)) is raised.

See also: `Size` ([279](#))

2.40.16 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` (275), `andBits` (275), `OrBits` (275), `XOrBits` (275)

2.40.17 TBits.SetIndex

Synopsis: Sets the start position for `FindNextBit` (278) and `FindPrevBit` (278)

Declaration: `procedure SetIndex(Index: LongInt)`

Visibility: public

Description: `SetIndex` sets the search start position for `FindNextBit` (278) and `FindPrevBit` (278) to `Index`. This means that these calls will start searching from position `Index`.

This mechanism provides an alternative to `FindFirstBit` (277) which can also be used to position for the `FindNextBit` and `FindPrevBit` calls.

Errors: None.

See also: `FindNextBit` (278), `FindPrevBit` (278), `FindFirstBit` (277), `OpenBit` (278)

2.40.18 TBits.FindFirstBit

Synopsis: Find first bit with a particular value

Declaration: `function FindFirstBit(State: Boolean) : LongInt`

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` (278) and `FindPrevBit` (278) 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` (277) instead.

Errors: None.

See also: `FindNextBit` (278), `FindPrevBit` (278), `OpenBit` (278), `SetIndex` (277)

2.40.19 TBits.FindNextBit

Synopsis: Searches the next bit with a particular value.

Declaration: `function FindNextBit : LongInt`

Visibility: `public`

Description: `FindNextBit` resumes a previously started search. It searches for the next bit with the value specified in the `FindFirstBit` (277). 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` (277).

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` (277), `FindPrevBit` (278), `OpenBit` (278), `SetIndex` (277)

2.40.20 TBits.FindPrevBit

Synopsis: Searches the previous bit with a particular value.

Declaration: `function FindPrevBit : LongInt`

Visibility: `public`

Description: `FindPrevBit` resumes a previously started search. It searches for the previous bit with the value specified in the `FindFirstBit` (277). 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` (277).

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` (277), `FindNextBit` (278), `OpenBit` (278), `SetIndex` (277)

2.40.21 TBits.OpenBit

Synopsis: Returns the position of the first bit that is set to `False`.

Declaration: `function OpenBit : LongInt`

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` (277), `FindPrevBit` (278), `FindFirstBit` (277), `SetIndex` (277)

2.40.22 TBits.Bits

Synopsis: Access to all bits in the array.

Declaration: `Property Bits[Bit: LongInt]: 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` (226) exception is raised.

See also: `Size` (279)

2.40.23 TBits.Size

Synopsis: Current size of the array of bits.

Declaration: `Property Size : LongInt`

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` (226) exception is raised.

See also: `Bits` (279)

2.41 TBytesStream

2.41.1 Description

`TBytesStream` is a stream that uses an array of byte (`TBytes` (195)) to keep the stream data. it overrides the `TMemoryStream` (339) memory allocation routine to use the array of bytes. The array of bytes is exposed through the `Bytes` (195) property.

See also: `TBytes` (195), `TMemoryStream` (339), `Bytes` (195)

2.41.2 Method overview

Page	Property	Description
280	<code>Create</code>	Create a new instance of the stream, initializing it with an array of bytes

2.41.3 Property overview

Page	Property	Access	Description
280	<code>Bytes</code>	<code>r</code>	The stream data as an array of bytes.

2.41.4 TBytesStream.Create

Synopsis: Create a new instance of the stream, initializing it with an array of bytes

Declaration: `constructor Create(const ABytes: TBytes);` Overload

Visibility: `public`

Description: `Create` creates a new instance and initializes the memory with the data in `ABytes`.

See also: `TBytes` ([195](#)), `TMemoryStream` ([339](#)), `Bytes` ([195](#))

2.41.5 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` ([304](#)), meaning that `Memory` points to the first byte in the array.

See also: `TBytes` ([195](#)), `TMemoryStream` ([339](#)), `TCustomMemoryStream.Memory` ([304](#))

2.42 TCollection

2.42.1 Description

`TCollection` implements functionality to manage a collection of named objects. Each of these objects needs to be a descendent of the `TCollectionItem` ([288](#)) class. Exactly which type of object is managed can be seen from the `TCollection.ItemClass` ([286](#)) property.

Normally, no `TCollection` is created directly. Instead, a descendent of `TCollection` and `TCollectionItem` ([288](#)) are created as a pair.

See also: `TCollectionItem` ([288](#))

2.42.2 Method overview

Page	Property	Description
282	Add	Creates and adds a new item to the collection.
282	Assign	Assigns one collection to another.
282	BeginUpdate	Start an update batch.
283	Clear	Removes all items from the collection.
281	Create	Creates a new collection.
283	Delete	Delete an item from the collection.
281	Destroy	Destroys the collection and frees all the objects it manages.
283	EndUpdate	Ends an update batch.
285	Exchange	Exchange 2 items in the collection
284	FindItemID	Searches for an Item in the collection, based on its TCollectionItem.ID (289) property.
284	GetEnumerator	Create an IEnumerator instance
284	GetNamePath	Overrides TPersistent.GetNamePath (352) to return a proper pathname.
284	Insert	Insert an item in the collection.
282	Owner	Owner of the collection.
285	Sort	Sort the items in the collection

2.42.3 Property overview

Page	Property	Access	Description
285	Count	r	Number of items in the collection.
286	ItemClass	r	Class pointer for each item in the collection.
286	Items	rw	Indexed array of items in the collection.

2.42.4 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` ([286](#)), `TCollection.Destroy` ([281](#))

2.42.5 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` ([281](#))

2.42.6 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.

2.42.7 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` (286) and adds it to the list. The newly created object is returned.

See also: `TCollection.ItemClass` (286), `TCollection.Clear` (283)

2.42.8 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` (351), `TCollectionItem` (288)

2.42.9 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` (280) 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` (283), even in case of an exception.

See also: `TCollection.EndUpdate` (283), `TCollection.Changed` (280), `TCollection.Update` (280)

2.42.10 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` (282), `TCollectionItem.Destroy` (288), `TCollection.Destroy` (281)

2.42.11 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` (280) it should always be used in conjunction with `TCollection.BeginUpdate` (282), preferably in the `Finally` section of a `Try ... Finally` block.

See also: `TCollection.BeginUpdate` (282), `TCollection.Changed` (280), `TCollection.Update` (280)

2.42.12 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` (286), `TCollection.Insert` (284), `TCollection.Clear` (283)

2.42.13 TCollection.GetEnumerator

Synopsis: Create an `IEnumerator` instance

Declaration: `function GetEnumerator : TCollectionEnumerator`

Visibility: public

Description: `GetEnumerator` is the implementation of the `IEnumerable` (1343) interface for `TCollection`. It creates a `TCollectionEnumerator` (286) instance and returns its `IEnumerator` (1343) interface.

See also: `TCollectionEnumerator` (286), `IEnumerator` (1343), `IEnumerable` (1343)

2.42.14 TCollection.GetNamePath

Synopsis: Overrides `TPersistent.GetNamePath` (352) 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` (280) 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` (280), `TCollection.Propname` (280)

2.42.15 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` (282) adds a new item at the end.

Errors: None.

See also: `TCollection.Add` (282), `TCollection.Delete` (283), `TCollection.Items` (286)

2.42.16 TCollection.FindItemID

Synopsis: Searches for an Item in the collection, based on its `TCollectionItem.ID` (289) 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` (289) 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` (286), `TCollectionItem.ID` (289)

2.42.17 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` (286), `TCollectionItem.Index` (290)

2.42.18 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` (200) type.

See also: `TCollectionSortCompare` (200)

2.42.19 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` (290) property, which is a zero-based index, meaning that it can take values between 0 and `Count-1`, borders included.

See also: `TCollectionItem.Index` (290), `TCollection.Items` (286)

2.42.20 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` (288), `TCollection.Items` (286)

2.42.21 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` (286) property.

Adding an element to the array can be done with the `TCollection.Add` (282) method. The array can be cleared with the `TCollection.Clear` (283) method. Removing an element of the array should be done by freeing that element.

See also: `TCollection.Count` (285), `TCollection.ItemClass` (286), `TCollection.Clear` (283), `TCollection.Add` (282)

2.43 TCollectionEnumerator

2.43.1 Description

`TCollectionEnumerator` implements the `#rtl.system.IEnumerator` (1343) interface for the `TCollection` (280) class, so the `TCollection` class can be used in a `for ... in` loop. It is returned by the `TCollection.GetEnumerator` (284) method of `TCollection`.

See also: `TCollection` (280), `TCollection.GetEnumerator` (284), `#rtl.system.IEnumerator` (1343)

2.43.2 Method overview

Page	Property	Description
287	Create	Initialize a new instance of <code>TCollectionEnumerator</code>
287	GetCurrent	Return the current pointer in the list
287	MoveNext	Move the position of the enumerator to the next position in the collection.

2.43.3 Property overview

Page	Property	Access	Description
287	Current	r	Current pointer in the list

2.43.4 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` ([280](#))

2.43.5 TCollectionEnumerator.GetCurrent

Synopsis: Return the current pointer in the list

Declaration: `function GetCurrent : TCollectionItem`

Visibility: `public`

Description: `GetCurrent` returns the current `TCollectionItem` ([288](#)) instance in the enumerator.

Errors: No checking is done on the validity of the current position.

See also: `MoveNext` ([287](#)), `TCollectionItem` ([288](#))

2.43.6 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` ([287](#))

2.43.7 TCollectionEnumerator.Current

Synopsis: Current pointer in the list

Declaration: `Property Current : TCollectionItem`

Visibility: `public`

Access: `Read`

Description: `Current` redefines `GetCurrent` ([287](#)) as a property.

See also: `GetCurrent` ([287](#))

2.44 TCollectionItem

2.44.1 Description

TCollectionItem and TCollection (280) 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 (290), TCollection.Index (280) and TCollectionItem.ID (289).

A TCollectionItem object is never created directly. To manage a set of named items, it is necessary to make a descendent of TCollectionItem to which needed properties and methods are added. This descendant can then be managed with a TCollection (280) class. The managing collection will create and destroy it's items by itself, it should therefore never be necessary to create TCollectionItem descendents manually.

See also: TCollection (280)

2.44.2 Method overview

Page	Property	Description
288	Create	Creates a new instance of this collection item.
288	Destroy	Destroys this collection item.
289	GetNamePath	Returns the namepath of this collection item.

2.44.3 Property overview

Page	Property	Access	Description
289	Collection	rw	Pointer to the collection managing this item.
290	DisplayName	rw	Name of the item, displayed in the object inspector.
289	ID	r	Initial index of this item.
290	Index	rw	Index of the item in its managing collection TCollection.Items (286) property.

2.44.4 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 (280). It is called by the TCollection.Add (282) function and should under normal circumstances never be called directly. called

See also: TCollectionItem.Destroy (288)

2.44.5 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` ([288](#))

2.44.6 `TCollectionItem.GetNamePath`

Synopsis: Returns the namepath of this collection item.

Declaration: `function GetNamePath : string; Override`

Visibility: public

Description: `GetNamePath` overrides the `TPersistent.GetNamePath` ([352](#)) method to return the name of the managing collection and appends its `TCollectionItem.Index` ([290](#)) property.

See also: `TCollectionItem.Collection` ([289](#)), `TPersistent.GetNamePath` ([352](#)), `TCollectionItem.Index` ([290](#))

2.44.7 `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` ([280](#))

2.44.8 `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` ([290](#)); 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` ([290](#)) 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 initialiy created, the `ID` and `Index` properties will be equal.

See also: `TCollection.Items` ([286](#)), `TCollectionItem.Index` ([290](#))

2.44.9 TCollectionItem.Index

Synopsis: Index of the item in its managing collection `TCollection.Items` (286) 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` (286) 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` (289) property does not form a continuous series, but can also be used to identify an item.

See also: `TCollectionItem.ID` (289), `TCollection.Items` (286)

2.44.10 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` (288) and `TCollectionItem.GetDisplayName` (288) to add storage functionality.

See also: `TCollectionItem.Index` (290), `TCollectionItem.ID` (289), `TCollectionItem.GetDisplayName` (288), `TCollectionItem.SetDisplayName` (288)

2.45 TComponent

2.45.1 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.

2.45.2 Interfaces overview

Page	Property	Description
233	IInterfaceComponentReference	Interface for checking component references
1345	IUnknown	Basic interface for all COM-based interfaces

2.45.3 Method overview

Page	Property	Description
293	BeforeDestruction	Overrides standard BeforeDestruction.
292	Create	Creates a new instance of the component.
293	Destroy	Destroys the instance of the component.
293	DestroyComponents	Destroy child components.
293	Destroying	Called when the component is being destroyed
294	ExecuteAction	Standard action execution method.
294	FindComponent	Finds and returns the named component in the owned components.
294	FreeNotification	Ask the component to notify called when it is being destroyed.
295	FreeOnRelease	Part of the IVCLComObject interface.
295	GetEnumerator	Create an IEnumerator instance
295	GetNamePath	Returns the name path of this component.
295	GetParentComponent	Returns the parent component.
296	HasParent	Does the component have a parent ?
296	InsertComponent	Insert the given component in the list of owned components.
297	IsImplementorOf	Checks if the current component is the implementor of the interface
292	Notification	Called by components that are freed and which received a FreeNotification.
297	ReferenceInterface	Interface implementation of Notification
296	RemoveComponent	Remove the given component from the list of owned components.
294	RemoveFreeNotification	Remove a component from the Free Notification list.
296	SafeCallException	Part of the IVCLComObject Interface.
297	SetSubComponent	Sets the csSubComponent style.
297	UpdateAction	Updates the state of an action.
292	WriteState	Writes the component to a stream.

2.45.4 Property overview

Page	Property	Access	Description
298	ComObject	r	Interface reference implemented by the component
298	ComponentCount	r	Count of owned components
298	ComponentIndex	rw	Index of component in it's owner's list.
298	Components	r	Indexed list (zero-based) of all owned components.
299	ComponentState	r	Current component's state.
299	ComponentStyle	r	Current component's style.
299	DesignInfo	rw	Information for IDE designer.
300	Name	rws	Name of the component.
300	Owner	r	Owner of this component.
300	Tag	rw	Tag value of the component.
300	VCLComObject	rw	Not implemented.

2.45.5 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` (294)) 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.

Descendents of `TComponent` can use `FreeNotification` (294) 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` (206), `TComponent.FreeNotification` (294), `TComponent.RemoveFreeNotification` (294)

2.45.6 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 (419) 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` (290) implementation of `WriteState` simply calls `TWriter.WriteData` (419). Descendent classes can, however, override `WriteState` to provide additional processing of stream data.

See also: `ReadState` (290), `TStream.WriteComponent` (373), `TWriter.WriteData` (419)

2.45.7 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` (290), `Owner` (300)

2.45.8 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` (293) (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 (292), Owner (300), `DestroyComponents` (293), Components (298)

2.45.9 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` (299)

See also: `TComponent.ComponentState` (299)

2.45.10 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 (298) 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` (293) destructor will not be called, which may result in an endless loop.

See also: `Destroy` (293), Components (298)

2.45.11 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 (290) property, and does the same for all owned components.

It is not necessary to call `Destroying` directly, the destructor `Destroy` (293) does this automatically.

See also: State (290), `Destroy` (293)

2.45.12 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` (255), `TBasicAction.ExecuteAction` (255), `TBasicAction.HandlesTarget` (256), `TComponent.UpdateAction` (297)

2.45.13 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` (298), `Name` (300)

2.45.14 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` (292) method is called for all components in the freenotification list.

See also: `Components` (298), `Notification` (292), `TComponent.RemoveFreeNotification` (294)

2.45.15 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` (294)

2.45.16 TComponent.FreeOnRelease

Synopsis: Part of the `IVCLComObject` interface.

Declaration: `procedure FreeOnRelease`

Visibility: `public`

Description: Provided for Delphi compatibility, but is not yet implemented.

2.45.17 TComponent.GetEnumerator

Synopsis: Create an `IEnumerator` instance

Declaration: `function GetEnumerator : TComponentEnumerator`

Visibility: `public`

Description: `GetEnumerator` is the implementation of the `IEnumerable` (1343) interface for `TComponent`. It creates a `TComponentEnumerator` (301) instance and returns its `IEnumerator` (1343) interface. The enumerator enumerates all child components of the component instance.

See also: `TComponentEnumerator` (301), `IEnumerator` (1343), `IEnumerable` (1343)

2.45.18 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` (300) property of the component.

See also: `Name` (300), `TPersistent.GetNamePath` (352)

2.45.19 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`. Descendent classes must override this method to return the visual parent of the component.

See also: `HasParent` (296), `Owner` (300)

2.45.20 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 descendent classes to return `True` when a parent is available. If `HasParent` returns `True`, then `GetParentComponent` (295) will return the parent component.

See also: `HasParent` (296), `Owner` (300)

2.45.21 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` (290) to see whether the component can be inserted. It then checks whether there are no name conflicts by calling `ValidateRename` (290). If neither of these checks have raised an exception the component is inserted, and notified of the insert.

See also: `RemoveComponent` (296), `Insert` (290), `ValidateContainer` (290), `ValidateRename` (290), `Notification` (292)

2.45.22 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` (296), `Remove` (290), `ValidateRename` (290), `Notification` (292)

2.45.23 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.

2.45.24 TComponent.SetSubComponent

Synopsis: Sets the `csSubComponent` style.

Declaration: `procedure SetSubComponent (ASubComponent: Boolean)`

Visibility: `public`

Description: `SetSubComponent` includes `csSubComponent` in the `ComponentStyle` (299) property if `ASubComponent` is `True`, and excludes it again if `ASubComponent` is `False`.

See also: `TComponent.ComponentStyle` (299)

2.45.25 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` (255), `TBasicAction.UpdateTarget` (257), `TBasicAction.HandlesTarget` (256), `TBasicAction.ExecuteAction` (255)

2.45.26 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` (233) and the `GetComponent` method should return the current instance.

See also: `IInterfaceComponentReference` (233)

2.45.27 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` (292) method of `TComponent` for interfaces. If the interface implements `IInterfaceComponentReference` (233), then the component that implements the interface is notified of the given operation `Op`.

Errors: None.

See also: `TComponent.Notification` (292), `IInterfaceComponentReference` (233)

2.45.28 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` (227).

See also: `EComponentError` (227)

2.45.29 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` (298).

See also: `ComponentCount` (298), `Owner` (300)

2.45.30 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` (298) array.

See also: `Components` (298), `Owner` (300)

2.45.31 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` (298), `ComponentCount` (298), `Owner` (300)

2.45.32 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 2.21: 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 its 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 free notifications.
<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` (290), `SetDesigning` (290), `SetInline` (290), `SetDesignInstance` (290), `Updating` (290), `Updated` (290), `Loaded` (290)

2.45.33 TComponent.ComponentStyle

Synopsis: Current component's style.

Declaration: `Property ComponentStyle : TComponentStyle`

Visibility: `public`

Access: `Read`

Description: Current component's style.

2.45.34 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` (300)

2.45.35 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` (296) method, or by removing the component from it's owner's owned component list using the `RemoveComponent` (296) method.

See also: `Components` (298), `InsertComponent` (296), `RemoveComponent` (296)

2.45.36 TComponent.VCLComObject

Synopsis: Not implemented.

Declaration: `Property VCLComObject : Pointer`

Visibility: `public`

Access: `Read,Write`

Description: `VCLComObject` is not yet implemented in Free Pascal.

2.45.37 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` (290), `Owner` (300)

2.45.38 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` (300)

2.46 TComponentEnumerator

2.46.1 Description

`TComponentEnumerator` implements the `#rtl.system.IEnumerator` (1343) interface for the `TComponent` (290) class, so the `TComponent` class can be used in a `for ... in` loop over the `TComponent.Components` (298) child components of the component. It is returned by the `TComponent.GetEnumerator` (295) method of `TComponent`.

See also: `TComponent` (290), `TComponent.GetEnumerator` (295), `#rtl.system.IEnumerator` (1343)

2.46.2 Method overview

Page	Property	Description
301	<code>Create</code>	Initialize a new instance of <code>TComponentEnumerator</code>
301	<code>GetCurrent</code>	Return the current pointer in the list
302	<code>MoveNext</code>	Move the position of the enumerator to the next position in the children of the component.

2.46.3 Property overview

Page	Property	Access	Description
302	<code>Current</code>	<code>r</code>	Current pointer in the list

2.46.4 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` (290)

2.46.5 TComponentEnumerator.GetCurrent

Synopsis: Return the current pointer in the list

Declaration: `function GetCurrent : TComponent`

Visibility: `public`

Description: `GetCurrent` returns the current `TComponent` (290) child component instance in the enumerator.

Errors: No checking is done on the validity of the current position.

See also: `MoveNext` (302), `TComponent.Components` (298)

2.46.6 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` ([301](#))

2.46.7 TComponentEnumerator.Current

Synopsis: Current pointer in the list

Declaration: `Property Current : TComponent`

Visibility: `public`

Access: `Read`

Description: `Current` redefines `GetCurrent` ([301](#)) as a property.

See also: `GetCurrent` ([301](#))

2.47 TCustomMemoryStream

2.47.1 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` ([304](#)) 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 descendent streams such as `TMemoryStream` ([339](#)). The reason for this approach is that this way it is possible to create e.g. read-only descendents 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 descendents such as `TMemoryStream` ([339](#)).

See also: `TMemoryStream` ([339](#)), `TStream` ([369](#))

2.47.2 Method overview

Page	Property	Description
303	<code>Read</code>	Reads <code>Count</code> bytes from the stream into <code>buffer</code> .
304	<code>SaveToFile</code>	Writes the contents of the stream to a file.
303	<code>SaveToStream</code>	Writes the contents of the memory stream to another stream.
303	<code>Seek</code>	Sets a new position in the stream.

2.47.3 Property overview

Page	Property	Access	Description
304	Memory	r	Pointer to the data kept in the memory stream.

2.47.4 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` ([370](#)) method of `TStream` ([369](#)). It will read as much bytes as are still available in the memory area pointer to by `Memory` ([304](#)). After the bytes are read, the internal stream position is updated.

See also: `TCustomMemoryStream.Memory` ([304](#)), `TStream.Read` ([370](#))

2.47.5 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` ([371](#)) 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` ([378](#)), `TStream.Size` ([378](#)), `TCustomMemoryStream.Memory` ([304](#))

2.47.6 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` ([372](#)) 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` ([229](#)) exception will be raised.

See also: `TCustomMemoryStream.SaveToFile` ([304](#)), `TStream.CopyFrom` ([372](#))

2.47.7 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` (303).

Remark: This method will work much faster than the use of the `TStream.CopyFrom` (372) 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` (229) exception may occur.

See also: `TCustomMemoryStream.SaveToStream` (303), `TFileStream` (310), `TStream.CopyFrom` (372)

2.47.8 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` (378)

2.48 TDataModule

2.48.1 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 descendents 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 descendent of `TDataModule` which contains non-visual components in it's published property list.

See also: `TDataModule.OnCreate` (307)

2.48.2 Method overview

Page	Property	Description
306	AfterConstruction	Overrides standard TObject (1349) behaviour.
306	BeforeDestruction	
305	Create	Create a new instance of a TDataModule.
305	CreateNew	
305	Destroy	Destroys the TDataModule instance.

2.48.3 Property overview

Page	Property	Access	Description
306	DesignOffset	rw	Position property needed for manipulation in an IDE.
307	DesignSize	rw	Size property needed for manipulation in an IDE.
307	OldCreateOrder	rw	Determines when OnCreate and OnDestroy are triggered.
307	OnCreate	rw	Event handler, called when the datamodule is created.
307	OnDestroy	rw	Event handler, called when the datamodule is destroyed.

2.48.4 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 ([305](#)). After that it reads the published properties from a stream using InitInheritedComponent ([218](#)) if a descendent class is instantiated. If the OldCreateOrder ([307](#)) property is `True`, the TDataModule.OnCreate ([307](#)) event is called.

Errors: An exception can be raised during the streaming operation.

See also: TDataModule.CreateNew ([305](#))

2.48.5 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 ([211](#)) handler is set, then it is called, with the newly created instance as an argument.

See also: TDataModule.Create ([305](#)), AddDataModule ([211](#)), TDataModule.OnCreate ([307](#))

2.48.6 TDataModule.Destroy

Synopsis: Destroys the TDataModule instance.

Declaration: `destructor Destroy; Override`

Visibility: public

Description: `Destroy` destroys the `TDataModule` instance. If the `OldCreateOrder` (307) property is `True` the `OnDestroy` (307) event handler is called prior to destroying the data module.

Before calling the inherited `destroy`, the `RemoveDataModule` (211) 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` (307), `RemoveDataModule` (211)

2.48.7 TDataModule.AfterConstruction

Synopsis: Overrides standard `TObject` (1349) behaviour.

Declaration: `procedure AfterConstruction; Override`

Visibility: public

Description: `AfterConstruction` calls the `OnCreate` (307) handler if the `OldCreateOrder` (307) property is `False`.

See also: `TDataModule.OldCreateOrder` (307), `TDataModule.OnCreate` (307)

2.48.8 TDataModule.BeforeDestruction

Synopsis:

Declaration: `procedure BeforeDestruction; Override`

Visibility: public

Description: `BeforeDestruction` calls the `OnDestroy` (307) handler if the `OldCreateOrder` (307) property is `False`.

See also: `TDataModule.OldCreateOrder` (307), `TDataModule.OnDestroy` (307)

2.48.9 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` (307)

2.48.10 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` ([306](#))

2.48.11 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` ([307](#)) property.

See also: `TDataModule.Create` ([305](#)), `TDataModule.CreateNew` ([305](#)), `TDataModule.OldCreateOrder` ([307](#))

2.48.12 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` ([307](#)) property.

See also: `TDataModule.Destroy` ([305](#)), `TDataModule.OnCreate` ([307](#)), `TDataModule.Create` ([305](#)), `TDataModule.CreateNew` ([305](#)), `TDataModule.OldCreateOrder` ([307](#))

2.48.13 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` (307) and `OnDestroy` (307) 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` (307), `TDataModule.OnCreate` (307), `TDataModule.Destroy` (305), `TDataModule.Create` (305), `TDataModule.CreateNew` (305), `TDataModule.OldCreateOrder` (307)

2.49 TFiler

2.49.1 Description

Class responsible for streaming of components.

2.49.2 Method overview

Page	Property	Description
308	<code>DefineBinaryProperty</code>	
308	<code>DefineProperty</code>	

2.49.3 Property overview

Page	Property	Access	Description
309	<code>Ancestor</code>	rw	Ancestor component from which an inherited component is streamed.
309	<code>IgnoreChildren</code>	rw	Determines whether children will be streamed as well.
309	<code>LookupRoot</code>	r	Component used to look up ancestor components.
309	<code>Root</code>	rw	The root component is the initial component which is being streamed.

2.49.4 TFiler.DefineProperty

Synopsis:

Declaration: `procedure DefineProperty(const Name: string; ReadData: TReaderProc; WriteData: TWriterProc; HasData: Boolean); Virtual; Abstract`

Visibility: `public`

Description:

2.49.5 TFiler.DefineBinaryProperty

Synopsis:

Declaration: `procedure DefineBinaryProperty(const Name: string; ReadData: TStreamProc; WriteData: TStreamProc; HasData: Boolean); Virtual; Abstract`

Visibility: public

Description:

2.49.6 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 \(309\)](#)

2.49.7 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` ([309](#)).

See also: [Root \(309\)](#)

2.49.8 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 \(309\)](#), [LookupRoot \(309\)](#)

2.49.9 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.

2.50 TFileStream

2.50.1 Description

`TFileStream` is a `TStream` (369) descendent that stores or reads it's data from a named file in the filesystem 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` (311) property to the list of public properties.

See also: `TFileStream.Create` (310), `TStream` (369)

2.50.2 Method overview

Page	Property	Description
310	Create	Creates a file stream.
310	Destroy	Destroys the file stream.

2.50.3 Property overview

Page	Property	Access	Description
311	FileName	r	The filename of the stream.

2.50.4 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 2.22:

<code>fmCreate</code>	<code>TFileStream.Create</code> (310) creates a new file if needed.
<code>fmOpenRead</code>	<code>TFileStream.Create</code> (310) opens a file with read-only access.
<code>fmOpenWrite</code>	<code>TFileStream.Create</code> (310) opens a file with write-only access.
<code>fmOpenReadWrite</code>	<code>TFileStream.Create</code> (310) opens a file with read-write access.

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 `EOpenError` (227) exception is raised.

See also: `TStream` (369), `TFileStream.FileName` (311), `THandleStream.Create` (321)

2.50.5 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` (310)

2.50.6 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` (310)

2.51 TFPList

2.51.1 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` (330), `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` (388) descendent such as `TStringList` (383). To manage general objects, a `TCollection` (280) class exists, from which a descendent can be made to manage collections of various kinds.

See also: `TStrings` (388), `TCollection` (280)

2.51.2 Method overview

Page	Property	Description
313	Add	Adds a new pointer to the list.
312	AddList	Add all pointers from another list
316	Assign	Assign performs the given operation on the list.
313	Clear	Clears the pointer list.
313	Delete	Removes a pointer from the list.
312	Destroy	Destroys the list and releases the memory used to store the list elements.
313	Error	Raises an <code>EListError</code> (228) exception.
314	Exchange	Exchanges two pointers in the list.
314	Expand	Increases the capacity of the list if needed.
314	Extract	Remove the first occurrence of a pointer from the list.
314	First	Returns the first non-nil pointer in the list.
317	ForEachCall	Call a procedure or method for each pointer in the list.
315	GetEnumerator	Create an <code>IEnumerator</code> instance
315	IndexOf	Returns the index of a given pointer.
315	IndexOfItem	Search an item in the list
315	Insert	Inserts a new pointer in the list at a given position.
316	Last	Returns the last non-nil pointer in the list.
316	Move	Moves a pointer from one position in the list to another.
317	Pack	Removes <code>Nil</code> pointers from the list and frees unused memory.
316	Remove	Removes a value from the list.
317	Sort	Sorts the pointers in the list.

2.51.3 Property overview

Page	Property	Access	Description
318	Capacity	rw	Current capacity (i.e. number of pointers that can be stored) of the list.
318	Count	rw	Current number of pointers in the list.
318	Items	rw	Provides access to the pointers in the list.
318	List	r	Memory array where pointers are stored.

2.51.4 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.

2.51.5 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` (316), `TList.AddList` (332)

2.51.6 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` (314) method.

To insert a pointer at a certain position in the list, use the `Insert` (315) method instead.

See also: `Delete` (313), `Grow` (330), `Insert` (315)

2.51.7 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` (312)

2.51.8 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.

2.51.9 TFPList.Error

Synopsis: Raises an `EListError` (228) exception.

Declaration: `class procedure Error(const Msg: string; Data: PtrInt)`

Visibility: `public`

Description: `Error` raises an `EListError` (228) exception, with a message formatted with `Msg` and `Data`.

2.51.10 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` (228) exception will be raised.

2.51.11 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` (318)

2.51.12 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` (313)

2.51.13 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` (316)

2.51.14 TFPList.GetEnumerator

Synopsis: Create an `IEnumerator` instance

Declaration: `function GetEnumerator : TFPListEnumerator`

Visibility: `public`

Description: `GetEnumerator` is the implementation of the `IEnumerable` (1343) interface for `TFPList`. It creates a `TFPListEnumerator` (319) instance and returns its `IEnumerator` (1343) interface.

See also: `TFPListEnumerator` (319), `IEnumerator` (1343), `IEnumerable` (1343)

2.51.15 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.

2.51.16 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` (195) 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` (315)

2.51.17 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` (228) exception is raised.

See also: `Add` (313), `Delete` (313)

2.51.18 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` ([314](#))

2.51.19 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` ([228](#)) exception is raised.

See also: `Exchange` ([314](#))

2.51.20 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` ([204](#)) type.

See also: `TFPList.Add` ([313](#)), `TFPList.Clear` ([313](#))

2.51.21 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` ([313](#)), `IndexOf` ([315](#)), `Insert` ([315](#))

2.51.22 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` ([313](#))

2.51.23 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.

2.51.24 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` ([205](#)), `TListCallback` ([205](#))

2.51.25 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` (313) or `insert` (315), 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` (311), `Count` (318)

2.51.26 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`.

2.51.27 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`.

2.51.28 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.

2.52 TFPListEnumerator

2.52.1 Description

`TFPListEnumerator` implements the `#rtl.system.IEnumerator` (1343) interface for the `TFPList` (311) class, so the `TFPList` class can be used in a `for ... in` loop. It is returned by the `TFPList.GetEnumerator` (315) method of `TFPList`.

See also: `TFPList` (311), `TFPList.GetEnumerator` (315), `#rtl.system.IEnumerator` (1343)

2.52.2 Method overview

Page	Property	Description
319	<code>Create</code>	Initialize a new instance of <code>TFPListEnumerator</code>
319	<code>GetCurrent</code>	Return the current pointer in the list
319	<code>MoveNext</code>	Move the position of the enumerator to the next position in the list.

2.52.3 Property overview

Page	Property	Access	Description
320	<code>Current</code>	<code>r</code>	Current pointer in the list

2.52.4 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` (311)

2.52.5 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` (319)

2.52.6 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` (319)

2.52.7 TFPListEnumerator.Current

Synopsis: Current pointer in the list

Declaration: `Property Current : Pointer`

Visibility: public

Access: Read

Description: `Current` redefines `GetCurrent` (319) as a property.

See also: `GetCurrent` (319)

2.53 THandleStream

2.53.1 Description

`THandleStream` is an abstract descendent of the `TStream` (369) 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` (321) and `Write` (321) 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; Descendent 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 descendent methods that support it.

See also: `TStream` (369), `TFileStream` (310)

2.53.2 Method overview

Page	Property	Description
321	Create	Create a handlestream from an OS Handle.
321	Read	Overrides standard read method.
321	Seek	Overrides the Seek method.
321	Write	Overrides standard write method.

2.53.3 Property overview

Page	Property	Access	Description
322	Handle	r	The OS handle of the stream.

2.53.4 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` (369)

2.53.5 THandleStream.Read

Synopsis: Overrides standard read method.

Declaration: `function Read(var Buffer; Count: LongInt) : LongInt; Override`

Visibility: public

Description: `Read` overrides the `Read` (370) method of `TStream`. It uses the `Handle` (322) 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` (370), `THandleStream.Write` (321), `THandleStream.Handle` (322)

2.53.6 THandleStream.Write

Synopsis: Overrides standard write method.

Declaration: `function Write(const Buffer; Count: LongInt) : LongInt; Override`

Visibility: public

Description: `Write` overrides the `Write` (370) method of `TStream`. It uses the `Handle` (322) 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` (370), `THandleStream.Write` (321), `THandleStream.Handle` (322)

2.53.7 THandleStream.Seek

Synopsis: Overrides the `Seek` method.

Declaration: `function Seek(const Offset: Int64; Origin: TSeekOrigin) : Int64
; Override`

Visibility: public

Description: `seek` uses the `FileSeek` (1438) method to position the stream on the desired position. Note that handle stream descendents (notably pipes) can override the method to prevent the seek.

2.53.8 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` (321)

See also: `THandleStream` (320), `THandleStream.Create` (321)

2.54 TInterfacedPersistent

2.54.1 Description

`TInterfacedPersistent` is a direct descendent of `TPersistent` (350) which implements the `#rtl.system.IInterface` (1150) interface. In particular, it implements the `QueryInterface` as a public method.

See also: `IInterface` (1150)

2.54.2 Interfaces overview

Page	Property	Description
1150	<code>IInterface</code>	Basic interface for all COM based interfaces

2.54.3 Method overview

Page	Property	Description
323	<code>AfterConstruction</code>	Overrides the standard <code>AfterConstruction</code> method.
322	<code>QueryInterface</code>	Implementation of <code>IInterface.QueryInterface</code>

2.54.4 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` (1357)

2.54.5 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).

2.55 TInterfaceList

2.55.1 Description

`TInterfaceList` is a standard implementation of the `IInterfaceList` (234) interface. It uses a `TThreadList` (416) instance to store the list of interfaces.

See also: `IInterfaceList` (234), `TList` (330)

2.55.2 Interfaces overview

Page	Property	Description
234	<code>IInterfaceList</code>	Interface for maintaining a list of interfaces.

2.55.3 Method overview

Page	Property	Description
326	<code>Add</code>	Add an interface to the list
324	<code>Clear</code>	Removes all interfaces from the list.
324	<code>Create</code>	Create a new instance of <code>TInterfaceList</code>
324	<code>Delete</code>	Delete an interface from the list.
324	<code>Destroy</code>	Destroys the list of interfaces
325	<code>Exchange</code>	Exchange 2 interfaces in the list
327	<code>Expand</code>	Expands the list
325	<code>First</code>	Returns the first non- <code>Nil</code> element in the list.
325	<code>GetEnumerator</code>	Create an <code>IEnumerator</code> instance
325	<code>IndexOf</code>	Returns the index of an interface.
326	<code>Insert</code>	Insert an interface to the list
326	<code>Last</code>	Returns the last non- <code>Nil</code> element in the list.
327	<code>Lock</code>	Lock the list
326	<code>Remove</code>	Remove an interface from the list
327	<code>Unlock</code>	UnLocks a locked list

2.55.4 Property overview

Page	Property	Access	Description
327	<code>Capacity</code>	<code>rw</code>	The current capacity of the list.
328	<code>Count</code>	<code>rw</code>	The current number of elements in the list.
328	<code>Items</code>	<code>rw</code>	Array-based access to the list's items.

2.55.5 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` ([324](#))

2.55.6 TInterfaceList.Destroy

Synopsis: Destroys the list of interfaces

Declaration: `destructor Destroy; Override`

Visibility: `public`

Description: `Destroy` first calls `Clear` ([324](#)) and then frees the `TInterfaceList` instance from memory.

Note that the `Clear` method decreases the reference count of all interfaces.

See also: `Create` ([324](#)), `Clear` ([324](#))

2.55.7 TInterfaceList.Clear

Synopsis: Removes all interfaces from the list.

Declaration: `procedure Clear`

Visibility: `public`

Description: `Clear` is the implementation of the `IInterfaceList.Clear` ([236](#)) 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` ([236](#)), `Add` ([326](#)), `Destroy` ([324](#)), `TList.Clear` ([333](#)), `TFPList.Clear` ([313](#))

2.55.8 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` ([236](#)) method. It clears the slot first and then removes the element from the list.

See also: `IInterfaceList.Delete` ([236](#)), `TInterfaceList.Remove` ([326](#)), `TInterfaceList.Add` ([326](#)), `TList.Delete` ([333](#)), `TFPList.Delete` ([313](#))

2.55.9 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` (237) method. It exchanges the position of 2 interfaces in the list.

See also: `IInterfaceList.Exchange` (237), `TInterfaceList.Delete` (324), `TInterfaceList.Add` (326), `TList.Exchange` (333), `TFPList.Exchange` (314)

2.55.10 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` (237) method. It returns the first non-`Nil` element from the list.

See also: `IInterfaceList.First` (237), `TList.First` (334)

2.55.11 TInterfaceList.GetEnumerator

Synopsis: Create an `IEnumerator` instance

Declaration: `function GetEnumerator : TInterfaceListEnumerator`

Visibility: public

Description: `GetEnumerator` is the implementation of the `IEnumerable` (1343) interface for `TInterfaceList`. It creates a `TInterfaceListEnumerator` (328) instance and returns its `IEnumerator` (1343) interface. The enumerator enumerates all interfaces in the list.

See also: `TInterfaceListEnumerator` (328), `IEnumerator` (1343), `IEnumerable` (1343)

2.55.12 TInterfaceList.IndexOf

Synopsis: Returns the index of an interface.

Declaration: `function IndexOf(item: IUnknown) : Integer`

Visibility: public

Description: `IndexOf` is the implementation of the `IInterfaceList.IndexOf` (237) 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` (237), `TList.IndexOf` (335)

2.55.13 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` (237) 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` (237), `TInterfaceList.Delete` (324), `TInterfaceList.Insert` (326), `TList.Add` (332), `TFPList.Add` (313)

2.55.14 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` (237) 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` (237), `TInterfaceList.Delete` (324), `TInterfaceList.Add` (326), `TList.Insert` (335), `TFPList.Insert` (315)

2.55.15 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` (238) method. It returns the last non-`Nil` element from the list.

See also: `IInterfaceList.Last` (238), `TInterfaceList.First` (325), `TList.Last` (335), `TFPList.Last` (316)

2.55.16 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` (238) method. It removes the first occurrence of the interface from the list.

See also: `IInterfaceList.Remove` (238), `TInterfaceList.Delete` (324), `TInterfaceList.IndexOf` (325), `TList.Remove` (336), `TFPList.Remove` (316)

2.55.17 TInterfaceList.Lock

Synopsis: Lock the list

Declaration: `procedure Lock`

Visibility: `public`

Description: `Lock` locks the list. It is the implementation of the `IInterfaceList.Lock` (238) method. It limits access to the list to the current thread.

See also: `IInterfaceList.Lock` (238), `TInterfaceList.Unlock` (327), `TThreadList.LockList` (418)

2.55.18 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` (238) method. After a call to unlock, the current thread releases the list for manipulation by other threads.

See also: `IInterfaceList.Unlock` (238), `TInterfaceList.Lock` (327), `TThreadList.UnlockList` (418)

2.55.19 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` (333)

2.55.20 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` (238), `TInterfaceList.Count` (328), `TList.Capacity` (337), `TFPList.Capacity` (318)

2.55.21 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` (239), `TInterfaceList.Items` (328), `TInterfaceList.Capacity` (327), `TList.Count` (337), `TFPList.Count` (318)

2.55.22 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` (328).

See also: `IInterfaceList.Items` (239), `TInterfaceList.Count` (328), `TList.Items` (337), `TFPList.Items` (318)

2.56 TInterfaceListEnumerator

2.56.1 Description

`TInterfaceListEnumerator` implements the `#rtl.system.IEnumerator` (1343) interface for the `TInterfaceList` (323) class, so the `TInterfaceList` class can be used in a `for ... in` loop over the `TInterfaceList.Components` (323) child components of the component. It is returned by the `TInterfaceList.GetEnumerator` (325) method of `TInterfaceList`.

See also: `TInterfaceList` (323), `TInterfaceList.GetEnumerator` (325), `#rtl.system.IEnumerator` (1343)

2.56.2 Method overview

Page	Property	Description
329	<code>Create</code>	Initialize a new instance of <code>TInterfaceListEnumerator</code>
329	<code>GetCurrent</code>	Return the current pointer in the list
329	<code>MoveNext</code>	Move the position of the enumerator to the next position in the children of the component.

2.56.3 Property overview

Page	Property	Access	Description
329	<code>Current</code>	<code>r</code>	Current pointer in the list

2.56.4 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` ([323](#))

2.56.5 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` ([323](#)) list.

Errors: No checking is done on the validity of the current position.

See also: `MoveNext` ([329](#)), `TInterfaceList.Components` ([323](#))

2.56.6 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` ([329](#))

2.56.7 TInterfaceListEnumerator.Current

Synopsis: Current pointer in the list

Declaration: `Property Current : IUnknown`

Visibility: `public`

Access: `Read`

Description: `Current` redefines `GetCurrent` ([329](#)) as a property.

See also: `GetCurrent` ([329](#))

2.57 TList

2.57.1 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` (311) may be used instead.

To manage collections of strings, it is better to use a `TStrings` (388) descendent such as `TStringList` (383). To manage general objects, a `TCollection` (280) class exists, from which a descendent can be made to manage collections of various kinds.

See also: `TStrings` (388), `TCollection` (280)

2.57.2 Interfaces overview

Page	Property	Description
231	<code>IFPObserved</code>	Interface implemented by an object that can be observed.

2.57.3 Method overview

Page	Property	Description
332	<code>Add</code>	Adds a new pointer to the list.
332	<code>AddList</code>	Add all pointers from another list
336	<code>Assign</code>	Copy the contents of other lists.
333	<code>Clear</code>	Clears the pointer list.
331	<code>Create</code>	Class to manage collections of pointers.
333	<code>Delete</code>	Removes a pointer from the list.
331	<code>Destroy</code>	Destroys the list and releases the memory used to store the list elements.
333	<code>Error</code>	Raises an <code>EListError</code> (228) exception.
333	<code>Exchange</code>	Exchanges two pointers in the list.
333	<code>Expand</code>	Increases the capacity of the list if needed.
334	<code>Extract</code>	Remove the first occurrence of a pointer from the list.
334	<code>First</code>	Returns the first non-nil pointer in the list.
331	<code>FPOAttachObserver</code>	Add an observer to the list of observers
331	<code>FPODetachObserver</code>	Remove an observer from the list of observers
332	<code>FPONotifyObservers</code>	Notify observers of changes in the list
334	<code>GetEnumerator</code>	Create an <code>IEnumerator</code> instance
335	<code>IndexOf</code>	Returns the index of a given pointer.
335	<code>Insert</code>	Inserts a new pointer in the list at a given position.
335	<code>Last</code>	Returns the last non-nil pointer in the list.
335	<code>Move</code>	Moves a pointer from one position in the list to another.
336	<code>Pack</code>	Removes <code>Nil</code> pointers from the list and frees unused memory.
336	<code>Remove</code>	Removes a value from the list.
336	<code>Sort</code>	Sorts the pointers in the list.

2.57.4 Property overview

Page	Property	Access	Description
337	Capacity	rw	Current capacity (i.e. number of pointers that can be stored) of the list.
337	Count	rw	Current number of pointers in the list.
337	Items	rw	Provides access to the pointers in the list.
338	List	r	Memory array where pointers are stored.

2.57.5 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` ([330](#)), `TList.Destroy` ([331](#))

2.57.6 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.

2.57.7 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` ([231](#)) 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` ([233](#)) interface.

See also: `IFPObserver` ([233](#)), `IFPObserved.FPOAttachObserver` ([232](#)), `IFPObserved` ([231](#))

2.57.8 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` (231) interface in `TList`. It removes the first found instance of the observer from the list of observers.

See also: `IFPObserved` (231), `IFPObserved.FPODetachObserver` (232), `IFPObserver` (233)

2.57.9 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` (195), `FPOAttachObserver` (195), `Add` (195), `Exchange` (195), `Delete` (195), `Extract` (195)

2.57.10 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` (336), `TFPList.AddList` (312)

2.57.11 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` (330) method.

To insert a pointer at a certain position in the list, use the `Insert` (335) method instead.

See also: `Delete` (333), `Grow` (330), `Insert` (335)

2.57.12 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` ([331](#))

2.57.13 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.

2.57.14 TList.Error

Synopsis: Raises an `EListError` ([228](#)) exception.

Declaration: `class procedure Error(const Msg: string; Data: PtrInt); Virtual`

Visibility: `public`

Description: `Error` raises an `EListError` ([228](#)) exception, with a message formatted with `Msg` and `Data`.

2.57.15 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` ([228](#)) exception will be raised.

2.57.16 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` ([337](#))

2.57.17 `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` ([333](#)), `TList.IndexOf` ([335](#)), `TList.Remove` ([336](#))

2.57.18 `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` ([335](#))

2.57.19 `TList.GetEnumerator`

Synopsis: Create an `IEnumerator` instance

Declaration: `function GetEnumerator : TListEnumerator`

Visibility: `public`

Description: `GetEnumerator` is the implementation of the `IEnumerable` ([1343](#)) interface for `TList`. It creates a `TListEnumerator` ([338](#)) instance and returns it's `IEnumerator` ([1343](#)) interface.

See also: `TListEnumerator` ([338](#)), `IEnumerator` ([1343](#)), `IEnumerable` ([1343](#))

2.57.20 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.

2.57.21 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` (228) exception is raised.

See also: `Add` (332), `Delete` (333)

2.57.22 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` (334)

2.57.23 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` (228) exception is raised.

See also: `Exchange` (333)

2.57.24 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` (204) type.

See also: `TList.Clear` (333)

2.57.25 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` (333), `IndexOf` (335), `Insert` (335)

2.57.26 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` (333)

2.57.27 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.

2.57.28 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` (332) or `insert` (335), 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` (330), `Count` (337)

2.57.29 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`.

2.57.30 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`.

2.57.31 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.

2.58 TListEnumerator

2.58.1 Description

`TListEnumerator` implements the `#rtl.system.IEnumerator` (1343) interface for the `TList` (330) class, so the `TList` class can be used in a `for ... in` loop. It is returned by the `TList.GetEnumerator` (334) method of `TList`.

See also: `TList` (330), `TList.GetEnumerator` (334), `#rtl.system.IEnumerator` (1343)

2.58.2 Method overview

Page	Property	Description
338	<code>Create</code>	Initialize a new instance of <code>TListEnumerator</code>
338	<code>GetCurrent</code>	Return the current pointer in the list
339	<code>MoveNext</code>	Move the position of the enumerator to the next position in the list.

2.58.3 Property overview

Page	Property	Access	Description
339	<code>Current</code>	<code>r</code>	Current pointer in the list

2.58.4 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` (330)

2.58.5 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` ([339](#))

2.58.6 `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` ([338](#))

2.58.7 `TListEnumerator.Current`

Synopsis: Current pointer in the list

Declaration: `Property Current : Pointer`

Visibility: `public`

Access: `Read`

Description: `Current` redefines `GetCurrent` ([338](#)) as a property.

See also: `GetCurrent` ([338](#))

2.59 `TMemoryStream`

2.59.1 Description

`TMemoryStream` is a `TStream` ([369](#)) descendent that stores its data in memory. It descends directly from `TCustomMemoryStream` ([302](#)) and implements the necessary to allocate and de-allocate memory directly from the heap. It implements the `Write` ([341](#)) 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` ([302](#)), `TStream` ([369](#))

2.59.2 Method overview

Page	Property	Description
340	Clear	Zeroes the position, capacity and size of the stream.
340	Destroy	Frees any allocated memory and destroys the memory stream.
341	LoadFromFile	Loads the contents of a file into memory.
340	LoadFromStream	Loads the contents of a stream into memory.
341	SetSize	Sets the size for the memory stream.
341	Write	Writes data to the stream's memory.

2.59.3 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.

2.59.4 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` ([378](#)), `TStream.Position` ([378](#)), `TCustomMemoryStream.Memory` ([304](#))

2.59.5 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` ([378](#)) 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` ([372](#)) 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` ([229](#)) may occur.

See also: `TStream.CopyFrom` ([372](#)), `TMemoryStream.LoadFromFile` ([341](#))

2.59.6 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` (340) method.

See also: `TMemoryStream.LoadFromStream` (340)

2.59.7 TMemoryStream.SetSize

Synopsis: Sets the size for the memory stream.

Declaration: `procedure SetSize(NewSize: LongInt); 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` (378), `TStream.Size` (378)

2.59.8 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` (370) method.

Errors: If no more memory could be allocated, then an exception will be raised.

See also: `TCustomMemoryStream.Read` (303)

2.60 TOwnedCollection

2.60.1 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` (280)

2.60.2 Method overview

Page	Property	Description
342	Create	Create a new <code>TOwnerCollection</code> instance.

2.60.3 `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` ([282](#)) 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` ([284](#)) and `Add` ([282](#)) methods.

See also: `TCollection.Create` ([281](#)), `TCollection.Owner` ([282](#))

2.61 `TOwnerStream`

2.61.1 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` ([343](#)) property is set to `True`).

See also: `TStream` ([369](#)), `TOwnerStream.Source` ([343](#)), `TOwnerStream.SourceOwner` ([343](#))

2.61.2 Method overview

Page	Property	Description
342	Create	Create a new instance of <code>TOwnerStream</code> .
343	Destroy	Destroys the <code>TOwnerStream</code> instance and the source stream.

2.61.3 Property overview

Page	Property	Access	Description
343	Source	r	Reference to the source stream.
343	SourceOwner	rw	Indicates whether the ownerstream owns it's source

2.61.4 `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 `AStream`. If `SourceOwner` is `True`, the source stream will also be freed when the instance is destroyed.

See also: `TOwnerStream.Destroy` ([343](#)), `TOwnerStream.Source` ([343](#)), `TOwnerStream.SourceOwner` ([343](#))

2.61.5 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` (342), `TOwnerStream.Source` (343), `TOwnerStream.SourceOwner` (343)

2.61.6 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 descendent 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` (342)

2.61.7 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` (343), `TOwnerStream.Destroy` (343)

2.62 TParser

2.62.1 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` (349) property. Various methods are provided to obtain the token value in the desired format.

To advance to the next token, invoke `NextToken` (346) method.

See also: `TParser.Token` (349), `TParser.NextToken` (346)

2.62.2 Method overview

Page	Property	Description
345	CheckToken	Checks whether the token is of the given type.
345	CheckTokenSymbol	Checks whether the token equals the given symbol
344	Create	Creates a new parser instance.
344	Destroy	Destroys the parser instance.
345	Error	Raises an <code>EParserError</code> (229) exception with the given message
345	ErrorFmt	Raises an <code>EParserError</code> (229) exception and formats the message.
345	ErrorStr	Raises an <code>EParserError</code> (229) exception with the given message
346	HexToBinary	Writes hexadecimal data to a stream.
346	NextToken	Reads the next token and returns its type.
346	SourcePos	Returns the current position in the stream.
347	TokenComponentIdent	Returns the path of a subcomponent starting from the current token.
347	TokenFloat	Returns the current token as a float.
347	TokenInt	Returns the current token as an integer.
348	TokenString	Returns the current token as a string.
348	TokenSymbolIs	Returns <code>True</code> if the token equals the given symbol.
348	TokenWideString	Returns the current token as a widestring

2.62.3 Property overview

Page	Property	Access	Description
349	FloatType	r	The type of a float token.
349	SourceLine	r	Current source line number.
349	Token	r	The type of the current token.

2.62.4 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` ([229](#)) exception is raised.

See also: `TParser.NextToken` ([346](#)), `TParser.Token` ([349](#))

2.62.5 TParser.Destroy

Synopsis: Destroys the parser instance.

Declaration: `destructor Destroy; Override`

Visibility: `public`

Description: Destroys the parser instance.

Errors: None.

2.62.6 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 `EParseError` (229) exception is raised.

See also: `TParser.Token` (349)

2.62.7 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` (197), otherwise an `EParseError` (229) exception is raised.

Errors: If the comparison fails, or current token isn't a symbol, an `EParseError` (229) exception is raised.

See also: `TParser.TokenSymbolIs` (348), `toSymbol` (197)

2.62.8 TParser.Error

Synopsis: Raises an `EParseError` (229) exception with the given message

Declaration: `procedure Error(const Ident: string)`

Visibility: public

Description: Raises an `EParseError` (229) exception with the given message

2.62.9 TParser.ErrorFmt

Synopsis: Raises an `EParseError` (229) exception and formats the message.

Declaration: `procedure ErrorFmt(const Ident: string; const Args: Array of const)`

Visibility: public

Description: Raises an `EParseError` (229) exception and formats the message.

2.62.10 TParser.ErrorStr

Synopsis: Raises an `EParseError` (229) exception with the given message

Declaration: `procedure ErrorStr(const Message: string)`

Visibility: public

Description: Raises an `EParseError` (229) exception with the given message

2.62.11 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` (229) exception is raised.

2.62.12 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` (349) property.

If the end of the stream is reached, `toEOF` (197) is returned.

For details about token types, see `TParser.Token` (349)

Errors: If an error occurs while parsing the token, an `EParseError` (229) exception is raised.

See also: `TParser.Token` (349)

2.62.13 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` (349)

2.62.14 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` (197), `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` (348) or `TokenWideString` (348) 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` (349) isn't `toSymbol` (197), or no valid symbol is found after a dot, an `EParserError` (229) exception is raised.

See also: `TParser.NextToken` (346), `TParser.Token` (349), `TParser.TokenString` (348), `TParser.TokenWideString` (348), `toSymbol` (197)

2.62.15 TParser.TokenFloat

Synopsis: Returns the current token as a float.

Declaration: `function TokenFloat : Extended`

Visibility: public

Description: If current token type is `toFloat` (197), 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` (349) for further information.

Remark: In the input stream the decimal separator, if present, must be a dot (.).

Errors: If `Token` (349) isn't `toFloat` (197), an `EParserError` (229) exception is raised.

See also: `TParser.FloatType` (349), `TParser.NextToken` (346), `TParser.Token` (349), `toFloat` (197)

2.62.16 TParser.TokenInt

Synopsis: Returns the current token as an integer.

Declaration: `function TokenInt : Int64`

Visibility: public

Description: If current token type is `toInteger` (197), 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` (349) isn't `toInteger` (197), an `EConvertError` (195) exception is raised.

See also: `TParser.NextToken` (346), `TParser.Token` (349), `toInteger` (197)

2.62.17 TParser.TokenString

Synopsis: Returns the current token as a string.

Declaration: `function TokenString : string`

Visibility: public

Description: If current token type is `toString` (197) or `toWString` (197), 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` (349).

Remark: If `Token` (349) is `toWString` (197), `TokenWideString` (348) should be used instead.

Errors: None.

See also: `TParser.NextToken` (346), `TParser.TokenWideString` (348), `TParser.Token` (349), `toString` (197), `toWString` (197)

2.62.18 TParser.TokenWideString

Synopsis: Returns the current token as a widestring

Declaration: `function TokenWideString : WideString`

Visibility: public

Description: If current token type is `toWString` (197), 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` (348).

Errors: None.

See also: `TParser.NextToken` (346), `TokenString` (348), `TParser.Token` (349), `toWString` (197)

2.62.19 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` (197), or comparison fails, `False` is returned.

Errors: None.

See also: `TParser.CheckTokenSymbol` (345), `TParser.Token` (349)

2.62.20 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 2.23:

s or S	Value is a single.
c or C	Value is a currency.
d or D	Value is a date.

If `Token` (349) isn't `toFloat` (197) or one of the above characters wasn't specified, `FloatType` is the null character (zero).

See also: `TParser.NextToken` (346), `TParser.Token` (349), `TParser.TokenFloat` (347), `toFloat` (197)

2.62.21 TParser.SourceLine

Synopsis: Current source line number.

Declaration: `Property SourceLine : Integer`

Visibility: `public`

Access: `Read`

Description: Current source line number.

See also: `TParser.SourcePos` (346)

2.62.22 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 2.24:

toEOF (197)	Value returned by <code>TParser.Token (349)</code> when the end of the input stream was reached.
toSymbol (197)	Value returned by <code>TParser.Token (349)</code> when a symbol was found in the input stream.
toString (197)	Value returned by <code>TParser.Token (349)</code> when a string was found in the input stream.
toInteger (197)	Value returned by <code>TParser.Token (349)</code> when an integer was found in the input stream.
toFloat (197)	Value returned by <code>TParser.Token (349)</code> when a floating point value was found in the input stream.
toWString (197)	Value returned by <code>TParser.Token (349)</code> when a wistring was found in the input stream.

To advance to the next token, use `NextToken (346)` method.

See also: `TParser.CheckToken (345)`, `TParser.NextToken (346)`, `TParser.TokenComponentIdent (347)`, `TParser.TokenFloat (347)`, `TParser.TokenInt (347)`, `TParser.TokenString (348)`, `TParser.TokenWideString (348)`

2.63 TPersistent

2.63.1 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 (231)` interface for the benefit of descendent classes, but does not call `IFPObserved.FPONotifyObservers (232)`. Descendents such as `TStrings (388)` and `TCollection (280)` and `TCollectionItem (288)` do use it.

See also: `TComponent (290)`, `IFPObserved (231)`, `TStrings (388)`, `TCollection (280)`, `TCollectionItem (288)`

2.63.2 Interfaces overview

Page	Property	Description
231	<code>IFPObserved</code>	Interface implemented by an object that can be observed.

2.63.3 Method overview

Page	Property	Description
351	<code>Assign</code>	Assign the contents of one class to another.
350	<code>Destroy</code>	Destroys the <code>TPersistent</code> instance.
351	<code>FPOAttachObserver</code>	Add an observer to the list of observers.
352	<code>FPODetachObserver</code>	Remove an observer from the list of observers
352	<code>FPONotifyObservers</code>	Notify observers of changes.
352	<code>GetNamePath</code>	Returns a string that can be used to identify the class instance.

2.63.4 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.

2.63.5 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` (350) 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` (350)

2.63.6 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` (231) 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` (233) interface.

See also: `IFPObserver` (233), `IFPObserved.FPOAttachObserver` (232), `IFPObserved` (231)

2.63.7 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` (231) interface in `TPersistent`. It removes the first found instance of the observer from the list of observers.

See also: `IFPObserved` (231), `IFPObserved.FPODetachObserver` (232), `IFPObserver` (233)

2.63.8 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` (233) interfaces.

`TPersistent` does not call `FPONotifyObservers`. It is implemented for the benefit of descendant classes.

See also: `IFPObserved` (231), `IFPObserved.FPONotifyObservers` (232), `IFPObserver` (233)

2.63.9 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` (350)

2.64 TProxyStream

2.64.1 Description

`TProxyStream` is a proxy class for the `#rtl.types.IStream` (1547) interface. It implements all stream methods by relaying them to the `IStream` interface.

See also: `#rtl.types.IStream` (1547), `TStreamAdapter` (379)

2.64.2 Method overview

Page	Property	Description
353	Check	Check errors
353	Create	Create a new instance of the TProxyStream class.
353	Read	
353	Seek	
353	Write	

2.64.3 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` ([1547](#))

2.64.4 TProxyStream.Read

Declaration: `function Read(var Buffer; Count: LongInt) : LongInt; Override`

Visibility: `public`

2.64.5 TProxyStream.Write

Declaration: `function Write(const Buffer; Count: LongInt) : LongInt; Override`

Visibility: `public`

2.64.6 TProxyStream.Seek

Declaration: `function Seek(const Offset: Int64; Origin: TSeekOrigin) : Int64; Override`

Visibility: `public`

2.64.7 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 descendent classes to return interface-specific errors.

See also: `#rtl.types.IStream` ([1547](#))

2.65 TReader

2.65.1 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` (243) 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` (372) call should be used.

See also: `TFile` (308), `TWriter` (419), `TAbstractObjectReader` (243)

2.65.2 Method overview

Page	Property	Description
356	BeginReferences	Initializes the component referencing mechanism.
357	CheckValue	Raises an exception if the next value in the stream is not of type Value
363	CopyValue	Copy a value to a writer.
356	Create	Creates a new reader class
357	DefineBinaryProperty	Reads a user-defined binary property from the stream.
357	DefineProperty	Reads a user-defined property from the stream.
356	Destroy	Destroys a reader class.
357	EndOfList	Returns true if the stream contains an end-of-list marker.
357	EndReferences	Finalizes the component referencing mechanism.
358	FixupReferences	Tries to resolve all unresolved component references.
358	NextValue	Returns the type of the next value.
358	Read	Read raw data from stream
358	ReadBoolean	Reads a boolean from the stream.
358	ReadChar	Reads a character from the stream.
359	ReadCollection	Reads a collection from the stream.
359	ReadComponent	Starts reading a component from the stream.
359	ReadComponents	Starts reading child components from the stream.
360	ReadCurrency	Read a currency value from the stream.
360	ReadDate	Reads a date from the stream
360	ReadFloat	Reads a float from the stream.
360	ReadIdent	Reads an identifier from the stream.
361	ReadInt64	Reads a 64-bit integer from the stream.
361	ReadInteger	Reads an integer from the stream
361	ReadListBegin	Checks for the beginning of a list.
361	ReadListEnd	Checks for the end of a list.
362	ReadRootComponent	Starts reading a root component.
361	ReadSet	Read a set value from the stream
360	ReadSingle	Reads a single-type real from the stream.
362	ReadString	Reads a string from the stream.
359	ReadUnicodeChar	Read unicode character
362	ReadUnicodeString	Read a UnicodeString value from the stream
363	ReadValue	Reads the next value type from the stream.
362	ReadVariant	Read a variant from the stream
359	ReadWideChar	Read widechar from the stream
362	ReadWideString	Read a WideString value from the stream.

2.65.3 Property overview

Page	Property	Access	Description
363	Driver	r	The driver in use for streaming the data.
365	OnAncestorNotFound	rw	Handler called when the ancestor component cannot be found.
365	OnCreateComponent	rw	Handler called when a component needs to be created.
364	OnError	rw	Handler called when an error occurs.
365	OnFindComponentClass	rw	Handler called when a component class reference needs to be found.
364	OnFindMethod	rw	Handler to find or change a method address.
364	OnPropertyNotFound	rw	Handler for treating missing properties.
366	OnReadStringProperty	rw	Handler for translating strings when read from the stream.
365	OnReferenceName	rw	Handler called when another component is referenced.
364	OnSetMethodProperty	rw	Handler for setting method properties.
365	OnSetName	rw	Handler called when setting a component name.
363	Owner	rw	Owner of the component being read
363	Parent	rw	Parent of the component being read.

2.65.4 TReader.Create

Synopsis: Creates a new reader class

Declaration: `constructor Create(Stream: TStream; BufSize: Integer)`

Visibility: `public`

Description: Creates a new reader class

2.65.5 TReader.Destroy

Synopsis: Destroys a reader class.

Declaration: `destructor Destroy; Override`

Visibility: `public`

Description: Destroys a reader class.

2.65.6 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.

2.65.7 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

2.65.8 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.

2.65.9 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.

2.65.10 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.

2.65.11 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.

2.65.12 TReader.FixupReferences

Synopsis: Tries to resolve all unresolved component references.

Declaration: `procedure FixupReferences`

Visibility: `public`

Description: Tries to resolve all unresolved component references.

2.65.13 TReader.NextValue

Synopsis: Returns the type of the next value.

Declaration: `function NextValue : TValueType`

Visibility: `public`

Description: Returns the type of the next value.

2.65.14 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` ([245](#)), `TBinaryObjectReader.Read` ([264](#))

2.65.15 TReader.ReadBoolean

Synopsis: Reads a boolean from the stream.

Declaration: `function ReadBoolean : Boolean`

Visibility: `public`

Description: Reads a boolean from the stream.

2.65.16 TReader.ReadChar

Synopsis: Reads a character from the stream.

Declaration: `function ReadChar : Char`

Visibility: `public`

Description: Reads a character from the stream.

2.65.17 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` ([362](#)), `TWriter.WriteWideChar` ([422](#))

2.65.18 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` ([362](#))

2.65.19 TReader.ReadCollection

Synopsis: Reads a collection from the stream.

Declaration: `procedure ReadCollection(Collection: TCollection)`

Visibility: public

Description: Reads a collection from the stream.

2.65.20 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.

2.65.21 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.

2.65.22 TReader.ReadFloat

Synopsis: Reads a float from the stream.

Declaration: `function ReadFloat : Extended`

Visibility: `public`

Description: Reads a float from the stream.

2.65.23 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.

2.65.24 TReader.ReadDate

Synopsis: Reads a date from the stream

Declaration: `function ReadDate : TDateTime`

Visibility: `public`

Description: Reads a date from the stream

2.65.25 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` ([423](#))

2.65.26 TReader.ReadIdent

Synopsis: Reads an identifier from the stream.

Declaration: `function ReadIdent : string`

Visibility: `public`

Description: Reads an identifier from the stream.

2.65.27 TReader.ReadInteger

Synopsis: Reads an integer from the stream

Declaration: `function ReadInteger : LongInt`

Visibility: `public`

Description: Reads an integer from the stream

2.65.28 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.

2.65.29 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` ([424](#))

2.65.30 TReader.ReadListBegin

Synopsis: Checks for the beginning of a list.

Declaration: `procedure ReadListBegin`

Visibility: `public`

Description: Checks for the beginning of a list.

2.65.31 TReader.ReadListEnd

Synopsis: Checks for the end of a list.

Declaration: `procedure ReadListEnd`

Visibility: `public`

Description: Checks for the end of a list.

2.65.32 TReader.ReadRootComponent

Synopsis: Starts reading a root component.

Declaration: `function ReadRootComponent (ARoot: TComponent) : TComponent`

Visibility: public

Description: Starts reading a root component.

2.65.33 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` ([272](#))

2.65.34 TReader.ReadString

Synopsis: Reads a string from the stream.

Declaration: `function ReadString : string`

Visibility: public

Description: Reads a string from the stream.

2.65.35 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` ([425](#))

2.65.36 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` ([250](#)), `TWriter.WriteString` ([425](#))

2.65.37 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.

2.65.38 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`.

2.65.39 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.

2.65.40 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

2.65.41 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.

2.65.42 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.

2.65.43 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 designtime by an IDE.

For more information about the meaning of the various arguments to the event handler, see `TPropertyNotFoundEvent` (206).

See also: `TPropertyNotFoundEvent` (206), `TReader.OnSetMethodProperty` (364), `TReader.OnReadStringProperty` (366)

2.65.44 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.

2.65.45 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` (366), `TReader.OnPropertyNotFound` (364)

2.65.46 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.

2.65.47 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.

2.65.48 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.

2.65.49 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.

2.65.50 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.

2.65.51 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` (207) for a description of the various parameters.

See also: `TReader.OnPropertyNotFound` (364), `TReader.OnSetMethodProperty` (364), `TReadWriteStringPropertyEvent` (207)

2.66 TRecall

2.66.1 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` (367) 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` (367) can be called.

See also: `TRecall.Create` (367), `TRecall.Destroy` (367), `TRecall.Forget` (367), `TRecall.Store` (367), `TPersistent.Assign` (351)

2.66.2 Method overview

Page	Property	Description
367	Create	Creates a new instance of <code>TRecall</code> .
367	Destroy	Copies the stored properties to the reference object and destroys the <code>TRecall</code> instance.
367	Forget	Clear the reference property.
367	Store	Assigns the reference instance to the storage instance.

2.66.3 Property overview

Page	Property	Access	Description
368	Reference	r	The reference object.

2.66.4 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` (367) to assign the reference object properties to the storage instance.

See also: TRecall.Store (367), TRecall.Destroy (367)

2.66.5 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 (368) instance is still valid. If the reference pointer was invalidated, call TRecall.Forget (367) to clear the reference instance.

See also: TRecall.Store (367), TRecall.Forget (367)

2.66.6 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 (368) instance is still valid. If the reference pointer was invalidated, call TRecall.Forget (367) to clear the reference instance.

2.66.7 TRecall.Forget

Synopsis: Clear the reference property.

Declaration: `procedure Forget`

Visibility: `public`

Description: `Forget` sets the Reference (368) 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` (367) has no effect.

Errors: None.

See also: TRecall.Reference (368), TRecall.Store (367), TRecall.Destroy (367)

2.66.8 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` (367) to clear the reference and then free the reference object.

See also: `TRecall.Forget` (367)

2.67 TResourceStream

2.67.1 Description

Stream that reads its data from a resource object.

2.67.2 Method overview

Page	Property	Description
368	<code>Create</code>	Creates a new instance of a resource stream.
368	<code>CreateFromID</code>	Creates a new instance of a resource stream with a resource
369	<code>Destroy</code>	Destroys the instance of the resource stream.

2.67.3 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.

2.67.4 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`.

2.67.5 TResourceStream.Destroy

Synopsis: Destroys the instance of the resource stream.

Declaration: `destructor Destroy; Override`

Visibility: `public`

Description: Destroys the instance of the resource stream.

2.68 TStream

2.68.1 Description

`TStream` is the base class for all streaming classes. It defines methods for reading (370), writing (370) from and to streams, as well as functions to determine the size of the stream as well as the current position of the stream.

Descendent classes such as `TMemoryStream` (339) or `TFileStream` (310) then override these methods to write streams to memory or file.

See also: `TMemoryStream` (339), `TFileStream` (310), `TStringStream` (405)

2.68.2 Method overview

Page	Property	Description
372	<code>CopyFrom</code>	Copy data from one stream to another
374	<code>FixupResourceHeader</code>	Not implemented in FPC
370	<code>Read</code>	Reads data from the stream to a buffer and returns the number of bytes read.
376	<code>ReadAnsiString</code>	Read an ansistring from the stream and return its value.
371	<code>ReadBuffer</code>	Reads data from the stream to a buffer
375	<code>ReadByte</code>	Read a byte from the stream and return its value.
372	<code>ReadComponent</code>	Reads component data from a stream
373	<code>ReadComponentRes</code>	Reads component data and resource header from a stream
376	<code>ReadDWord</code>	Read a DWord from the stream and return its value.
376	<code>ReadQWord</code>	Read a QWord value from the stream and return its value
375	<code>ReadResHeader</code>	Read a resource header from the stream.
375	<code>ReadWord</code>	Read a word from the stream and return its value.
371	<code>Seek</code>	Sets the current position in the stream
370	<code>Write</code>	Writes data from a buffer to the stream and returns the number of bytes written.
378	<code>WriteAnsiString</code>	Write an ansistring to the stream.
372	<code>WriteBuffer</code>	Writes data from a buffer to the stream
376	<code>WriteByte</code>	Write a byte to the stream.
373	<code>WriteComponent</code>	Write component data to the stream
373	<code>WriteComponentRes</code>	Write resource header and component data to a stream
374	<code>WriteDescendent</code>	Write component data to a stream, relative to an ancestor
374	<code>WriteDescendentRes</code>	Write resource header and component data to a stream, relative to an ancestor
377	<code>WriteDWord</code>	Write a DWord to the stream.
377	<code>WriteQWord</code>	Write a QWord value to the stream
374	<code>WriteResourceHeader</code>	Write resource header to the stream
377	<code>WriteWord</code>	Write a word to the stream.

2.68.3 Property overview

Page	Property	Access	Description
378	Position	rw	The current position in the stream.
378	Size	rw	The current size of the stream.

2.68.4 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`

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` ([371](#)) instead.

As implemented in `TStream`, `Read` does nothing but raises an `EStreamError` ([229](#)) exception to indicate that reading is not supported. Descendent classes that allow reading must override this method to do the actual reading.

Descendent classes should (if they don't explicitly raise an exception) return a positive value (≥ 0), where zero indicates an error.

Errors: In case a descendent class does not allow reading from the stream, an exception is raised.

See also: `TStream.Write` ([370](#)), `TStream.ReadBuffer` ([371](#))

2.68.5 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`

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` ([372](#)) instead.

As implemented in `TStream`, `Write` does nothing but raises `EStreamError` ([229](#)) exception to indicate that writing is not supported. Descendent classes that allow writing must override this method to do the actual writing.

Descendent classes should (if they don't explicitly raise an exception) return a positive value (≥ 0), where zero indicates an error.

Errors: In case a descendent class does not allow writing to the stream, an exception is raised.

See also: `TStream.Read` ([370](#)), `TStream.WriteBuffer` ([372](#))

2.68.6 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` (207) selects the 64-bit variant of the function.

The `Origin` parameter for the 32-bit version can have one of the following values:

Table 2.25:

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` (195) unit.

The `Origin` parameter for the 64-bit version has one of the following values:

Table 2.26:

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 descendent 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 descendent 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` (378)

2.68.7 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` (370) 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` (370), `TStream.WriteBuffer` (372)

2.68.8 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` (370) 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` (370), `TStream.ReadBuffer` (371)

2.68.9 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` (370), `TStream.Write` (370)

2.68.10 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` (354) object and calls its `ReadRootComponent` (362) method.

Errors: If an error occurs during the reading of the component, an `EFileError` (227) exception is raised.

See also: `TStream.WriteComponent` (373), `TStream.ReadComponentRes` (373), `TReader.ReadRootComponent` (362)

2.68.11 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` (372) 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` (227) exception is raised.

See also: `TStream.ReadComponent` (372), `TStream.WriteComponentRes` (373)

2.68.12 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` (372). 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 `WriteDescendent` (374) with `Nil` ancestor.

See also: `TStream.ReadComponent` (372), `TStream.WriteComponentRes` (373)

2.68.13 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` (373) 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` (373), `TStream.ReadComponentRes` (373)

2.68.14 TStream.WriteDescendent

Synopsis: Write component data to a stream, relative to an ancestor

Declaration: `procedure WriteDescendent (Instance: TComponent; Ancestor: TComponent)`

Visibility: public

Description: `WriteDescendent` writes the state of `Instance` to the stream where it differs from `Ancestor`, i.e. only the changed properties are written to the stream.

`WriteDescendent` creates a `TWriter` (419) object and calls its `WriteDescendent` (422) object. The writer is passed a binary driver object (268) by default.

2.68.15 TStream.WriteDescendentRes

Synopsis: Write resource header and component data to a stream, relative to an ancestor

Declaration: `procedure WriteDescendentRes (const ResName: string; Instance: TComponent; Ancestor: TComponent)`

Visibility: public

Description: `WriteDescendentRes` writes a `ResName` resource header, and then calls `WriteDescendent` (374) to write the state of `Instance` to the stream where it differs from `Ancestor`, i.e. only the changed properties are written to the stream.

This method is intened for use by an IDE that can use it to store forms or datamodules as designed in a Windows resource stream.

2.68.16 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` (374).

`WriteResourceHeader` should not be used directly. It is called by the `TStream.WriteComponentRes` (373) and `TStream.WriteDescendentRes` (374) methods.

See also: `TStream.FixupResourceHeader` (374), `TStream.WriteComponentRes` (373), `TStream.WriteDescendentRes` (374)

2.68.17 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 \(374\)](#), [TStream.WriteComponentRes \(373\)](#), [TStream.WriteDescendentRes \(374\)](#)

2.68.18 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 \(228\)](#) exception is raised.

See also: [TStream.ReadComponentRes \(373\)](#), [EInvalidImage \(228\)](#)

2.68.19 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, an [EStreamError \(229\)](#) exception will be raised. This is a utility function which simply calls the [Read \(370\)](#) function.

See also: [TStream.Read \(370\)](#), [TStream.WriteByte \(376\)](#), [TStream.ReadWord \(375\)](#), [TStream.ReadDWord \(376\)](#), [TStream.ReadAnsiString \(376\)](#)

2.68.20 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 \(370\)](#) function.

Errors: If the word cannot be read, an [EStreamError \(229\)](#) exception will be raised.

See also: [TStream.Read \(370\)](#), [TStream.WriteWord \(377\)](#), [TStream.ReadByte \(375\)](#), [TStream.ReadDWord \(376\)](#), [TStream.ReadAnsiString \(376\)](#)

2.68.21 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` (370) function.

Errors: If the DWord cannot be read, a `EStreamError` (229) exception will be raised.

See also: `TStream.Read` (370), `TStream.WriteDWord` (377), `TStream.ReadByte` (375), `TStream.ReadWord` (375), `TStream.ReadAnsiString` (376)

2.68.22 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` (229) exception will be raised.

See also: `TStream.Read` (370), `TStream.WriteByte` (376), `TStream.ReadWord` (375), `TStream.ReadDWord` (376), `TStream.ReadAnsiString` (376)

2.68.23 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` (378) function writes an ansistring in such a format.

Errors: If the AnsiString cannot be read, a `EStreamError` (229) exception will be raised.

See also: `TStream.Read` (370), `TStream.WriteAnsiString` (378), `TStream.ReadByte` (375), `TStream.ReadWord` (375), `TStream.ReadDWord` (376)

2.68.24 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` (370) function. The byte can be read from the stream using the `ReadByte` (375) function.

Errors: If an error occurs when attempting to write, an `EStreamError` (229) exception will be raised.

See also: `TStream.Write` (370), `TStream.ReadByte` (375), `TStream.WriteWord` (377), `TStream.WriteDWord` (377), `TStream.WriteAnsiString` (378)

2.68.25 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` (370) function. The word can be read from the stream using the `ReadWord` (375) function.

Errors: If an error occurs when attempting to write, an `EStreamError` (229) exception will be raised.

See also: `TStream.Write` (370), `TStream.ReadWord` (375), `TStream.WriteByte` (376), `TStream.WriteDWord` (377), `TStream.WriteAnsiString` (378)

2.68.26 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` (370) function. The DWord can be read from the stream using the `ReadDWord` (376) function.

Errors: If an error occurs when attempting to write, an `EStreamError` (229) exception will be raised.

See also: `TStream.Write` (370), `TStream.ReadDWord` (376), `TStream.WriteByte` (376), `TStream.WriteWord` (377), `TStream.WriteAnsiString` (378)

2.68.27 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` (370) function. The word can be read from the stream using the `ReadQWord` (376) function.

Errors: If an error occurs when attempting to write, an `EStreamError` (229) exception will be raised.

See also: `TStream.Write` (370), `TStream.ReadByte` (375), `TStream.WriteWord` (377), `TStream.WriteDWord` (377), `TStream.WriteAnsiString` (378)

2.68.28 TStream.WriteAnsiString

Synopsis: Write an ansistring to the stream.

Declaration: `procedure WriteAnsiString(const S: string)`

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` (370) 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` (376) function.

Errors: If an error occurs when attempting to write, an `EStreamError` (229) exception will be raised.

See also: `TStream.Write` (370), `TStream.ReadAnsiString` (376), `TStream.WriteByte` (376), `TStream.WriteWord` (377), `TStream.WriteDWord` (377)

2.68.29 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` (229) exception if they do not support setting the stream position.

See also: `TStream.Size` (378), `TStream.Seek` (371)

2.68.30 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` (378), `TStream.Seek` (371)

2.69 TStreamAdapter

2.69.1 Description

Implements IStream for TStream (369) descendents

2.69.2 Interfaces overview

Page	Property	Description
1547	IStream	COM stream abstraction

2.69.3 Method overview

Page	Property	Description
382	Clone	Clone the stream
381	Commit	Commit data to the stream
381	CopyTo	Copy data to destination stream
379	Create	Create a new instance of TStreamAdapter
379	Destroy	Free the TStreamAdapter instance
382	LockRegion	Lock a region of the stream
380	Read	Read from the stream.
381	Revert	Revert operations on the stream
380	Seek	Set the stream position
381	SetSize	Set the stream size
382	Stat	Return statistical data about the stream
382	UnlockRegion	Unlock a region of the stream
380	Write	Write to the stream

2.69.4 Property overview

Page	Property	Access	Description
383	Stream	r	Stream on which adaptor works
383	StreamOwnership	rw	Determines what happens with the stream when the adaptor is freed

2.69.5 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 TStreamAdaptor. It initializes TStreamAdapter.Stream (383) with Stream and initializes StreamOwnership (383) with Ownership.

TStreamAdapter is an abstract class: descendents must be created that implement the actual functionality.

See also: StreamOwnership (383), TStreamAdapter.Stream (383)

2.69.6 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.

2.69.7 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` ([1547](#)) 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` ([1547](#))

2.69.8 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` ([1547](#)) 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` ([1547](#))

2.69.9 TStreamAdapter.Seek

Synopsis: Set the stream position

Declaration: function `Seek`(`dlibMove: Largeint`; `dwOrigin: LongInt`;
out `libNewPosition: Largeint`) : `HRESULT`; `Virtual`

Visibility: `public`

Description: `Seek` implements `#rtl.types.IStream.Seek` ([1548](#)) 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` ([1548](#))

2.69.10 TStreamAdapter.SetSize

Synopsis: Set the stream size

Declaration: `function SetSize(libNewSize: Largeint) : HRESULT; Virtual`

Visibility: public

Description: `SetSize` implements `#rtl.types.IStream.SetSize` (1548) 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` (1548)

2.69.11 TStreamAdapter.CopyTo

Synopsis: Copy data to destination stream

Declaration: `function CopyTo(stm: IStream;cb: Largeint;out cbRead: Largeint;
out cbWritten: Largeint) : HRESULT; Virtual`

Visibility: public

Description: `CopyTo` implements `#rtl.types.IStream.CopyTo` (1548).

Errors: This function must be overridden and will raise a runerror 217 when called directly.

2.69.12 TStreamAdapter.Commit

Synopsis: Commit data to the stream

Declaration: `function Commit(grfCommitFlags: LongInt) : HRESULT; Virtual`

Visibility: public

Description: `Commit` implements `#rtl.types.IStream.Commit` (1549).

Errors: This function must be overridden and will raise a runerror 217 when called directly.

See also: `#rtl.types.IStream.Commit` (1549)

2.69.13 TStreamAdapter.Revert

Synopsis: Revert operations on the stream

Declaration: `function Revert : HRESULT; Virtual`

Visibility: public

Description: `Revert` implements `#rtl.types.IStream.Revert` (1549).

Errors: This function must be overridden and will raise a runerror 217 when called directly.

See also: `#rtl.types.IStream.Revert` (1549)

2.69.14 TStreamAdapter.LockRegion

Synopsis: Lock a region of the stream

Declaration: `function LockRegion(libOffset: Largeint;cb: Largeint;
dwLockType: LongInt) : HRESULT; Virtual`

Visibility: public

Description: `LockRegion` implements `#rtl.types.IStream.LockRegion` (1549).

Errors: This function must be overridden and will raise a `runerror 217` when called directly.

See also: `#rtl.types.IStream.LockRegion` (1549)

2.69.15 TStreamAdapter.UnlockRegion

Synopsis: Unlock a region of the stream

Declaration: `function UnlockRegion(libOffset: Largeint;cb: Largeint;
dwLockType: LongInt) : HRESULT; Virtual`

Visibility: public

Description: `UnLockRegion` implements `#rtl.types.IStream.UnLockRegion` (1549).

Errors: This function must be overridden and will raise a `runerror 217` when called directly.

See also: `#rtl.types.IStream.UnLockRegion` (1549)

2.69.16 TStreamAdapter.Stat

Synopsis: Return statistical data about the stream

Declaration: `function Stat(out statstg: TStatStg;grfStatFlag: LongInt) : HRESULT
; Virtual`

Visibility: public

Description: `Stat` implements `#rtl.types.IStream.Stat` (1550).

Errors: This function must be overridden and will raise a `runerror 217` when called directly.

See also: `#rtl.types.IStream.Stat` (1550)

2.69.17 TStreamAdapter.Clone

Synopsis: Clone the stream

Declaration: `function Clone(out stm: IStream) : HRESULT; Virtual`

Visibility: public

Description: `Clone` implements `#rtl.types.IStream.Clone` (1550).

Errors: This function must be overridden and will raise a `runerror 217` when called directly.

See also: `#rtl.types.IStream.Clone` (1550)

2.69.18 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.

2.69.19 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

2.70 TStringList

2.70.1 Description

`TStringList` is a descendent class of `TStrings` (388) 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` (388), `TStringList.Duplicates` (386), `TStringList.Sorted` (387)

2.70.2 Method overview

Page	Property	Description
384	Add	Implements the <code>TStrings.Add</code> (390) function.
384	Clear	Implements the <code>TStrings.Clear</code> (392) function.
386	CustomSort	Sort the stringlist using a custom sort algorithm
385	Delete	Implements the <code>TStrings.Delete</code> (392) function.
384	Destroy	Destroys the stringlist.
385	Exchange	Implements the <code>TStrings.Exchange</code> (393) function.
385	Find	Locates the index for a given string in sorted lists.
385	IndexOf	Overrides the <code>TStrings.IndexOf</code> (394) property.
386	Insert	Overrides the <code>TStrings.Insert</code> (395) method.
386	Sort	Sorts the strings in the list.

2.70.3 Property overview

Page	Property	Access	Description
387	CaseSensitive	rw	
386	Duplicates	rw	Describes the behaviour of a sorted list with respect to duplicate strings.
387	OnChange	rw	Event triggered after the list was modified.
388	OnChanging	rw	Event triggered when the list is about to be modified.
388	OwnsObjects	rw	Determines whether the stringlist owns its objects or not.
387	Sorted	rw	Determines whether the list is sorted or not.

2.70.4 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.

2.70.5 TStringList.Add

Synopsis: Implements the `TStrings.Add` ([390](#)) 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` ([386](#)) is `dupError` then an `EStringListError` ([230](#)) 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` ([386](#)) is `dupError` then an `EStringListError` ([230](#)) exception is raised.

See also: `TStringList.Insert` ([386](#)), `TStringList.Duplicates` ([386](#))

2.70.6 TStringList.Clear

Synopsis: Implements the `TStrings.Clear` ([392](#)) function.

Declaration: `procedure Clear; Override`

Visibility: `public`

Description: Implements the `TStrings.Clear` ([392](#)) function.

2.70.7 TStringList.Delete

Synopsis: Implements the TStrings.Delete ([392](#)) function.

Declaration: `procedure Delete(Index: Integer); Override`

Visibility: `public`

Description: Implements the TStrings.Delete ([392](#)) function.

2.70.8 TStringList.Exchange

Synopsis: Implements the TStrings.Exchange ([393](#)) 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 ([393](#)).

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 ([387](#)), TStrings.Exchange ([393](#))

2.70.9 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 ([385](#)) instead.
2. Find uses a binary search method to locate the string

2.70.10 TStringList.IndexOf

Synopsis: Overrides the TStrings.IndexOf ([394](#)) property.

Declaration: `function IndexOf(const S: string) : Integer; Override`

Visibility: `public`

Description: IndexOf overrides the ancestor method TStrings.IndexOf ([394](#)). 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 ([394](#)), TStringList.Find ([385](#))

2.70.11 TStringList.Insert

Synopsis: Overrides the TStrings.Insert (395) 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` (230) 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` (230) exception will be raised.

See also: `TStringList.Add` (384), `TStrings.Insert` (395), `TStrings.InsertObject` (395)

2.70.12 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` (387) property set to `True` then nothing will be done.

See also: `TStringList.Sorted` (387)

2.70.13 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` (387), `TStringList.Sort` (386)

2.70.14 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 2.27:

<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> (230) exception is raised.
<code>dupAccept</code>	Duplicate values can be added to the list.

If the stringlist is not sorted, the `Duplicates` setting is ignored.

2.70.15 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` (386) cannot be used. For sorted lists, `TStringList.Add` (384) should be used instead.
2. If `Sorted` is `True`, the `TStringList.Duplicates` (386) setting has effect. This setting is ignored when `Sorted` is `False`.

See also: `TStringList.Sort` (386), `TStringList.Duplicates` (386), `TStringList.Add` (384), `TstringList.Insert` (386)

2.70.16 TStringList.CaseSensitive

Synopsis:

Declaration: `Property CaseSensitive : Boolean`

Visibility: `public`

Access: `Read, Write`

Description: Indicates whether locating strings happens in a case sensitive manner.

2.70.17 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` (388) event occurs.

See also: `TStringList.OnChanging` (388)

2.70.18 `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` (387) event occurs.

See also: `TStringList.OnChange` (387)

2.70.19 `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` (402)

2.71 `TStrings`

2.71.1 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 descendent class such as `TStringList` (383) 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 descendents such as `TStringList` (383).

`TStrings` implements the `IFPObserved` (231) interface: when the stringlist is changed, a `ooChanged` notification is sent to all observers.

See also: [TStringList \(383\)](#), [IFPObserved \(231\)](#)

2.71.2 Method overview

Page	Property	Description
390	Add	Add a string to the list
390	AddObject	Add a string and associated object to the list.
391	AddStrings	Add contents of another stringlist to this list.
391	AddText	Add text to the string list.
391	Append	Add a string to the list.
391	Assign	Assign the contents of another stringlist to this one.
392	BeginUpdate	Mark the beginning of an update batch.
392	Clear	Removes all strings and associated objects from the list.
392	Delete	Delete a string from the list.
390	Destroy	Frees all strings and objects, and removes the list from memory.
393	EndUpdate	Mark the end of an update batch.
393	Equals	Compares the contents of two stringlists.
393	Exchange	Exchanges two strings in the list.
398	ExtractName	Extract the name part of a string
394	GetEnumerator	Create an <code>IEnumerator</code> instance
398	GetNameValue	Return both name and value of a name,value pair based on it's index.
394	GetText	Returns the contents as a <code>PChar</code>
394	IndexOf	Find a string in the list and return its position.
394	IndexOfName	Finds the index of a name in the name-value pairs.
395	IndexOfObject	Finds an object in the list and returns its index.
395	Insert	Insert a string in the list.
395	InsertObject	Insert a string and associated object in the list.
396	LoadFromFile	Load the contents of a file as a series of strings.
396	LoadFromStream	Load the contents of a stream as a series of strings.
396	Move	Move a string from one place in the list to another.
397	SaveToFile	Save the contents of the list to a file.
397	SaveToStream	Save the contents of the string to a stream.
397	SetText	Set the contents of the list from a <code>PChar</code> .

2.71.3 Property overview

Page	Property	Access	Description
400	Capacity	rw	Capacity of the list, i.e. number of strings that the list can currently hold before it tries to expand.
400	CommaText	rw	Contents of the list as a comma-separated string.
401	Count	r	Number of strings in the list.
399	DelimitedText	rw	Get or set all strings in the list in a delimited form.
398	Delimiter	rw	Delimiter character used in DelimitedText (399).
402	Names	r	Name parts of the name-value pairs in the list.
400	NameValueSeparator	rw	Value of the character used to separate name,value pairs
402	Objects	rw	Indexed access to the objects associated with the strings in the list.
399	QuoteChar	rw	Quote character used in DelimitedText (399).
399	StrictDelimiter	rw	Should only the delimiter character be considered a delimiter
403	Strings	rw	Indexed access to the strings in the list.
404	StringsAdapter	rw	Not implemented in Free Pascal.
403	Text	rw	Contents of the list as one big string.
398	TextLineBreakStyle	rw	Determines which line breaks to use in the Text (403) property
400	ValueFromIndex	rw	Return the value part of a string based on it's index.
402	Values	rw	Value parts of the name-value pairs in the list.

2.71.4 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.

2.71.5 TStrings.Add

Synopsis: Add a string to the list

Declaration: `function Add(const S: string) : Integer; Virtual`

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` ([401](#)))

See also: `TStrings.Objects` ([402](#)), `TStrings.AddObject` ([390](#)), `TStrings.Insert` ([395](#)), `TStrings.Delete` ([392](#)), `TStrings.Strings` ([403](#)), `TStrings.Count` ([401](#))

2.71.6 TStrings.AddObject

Synopsis: Add a string and associated object to the list.

Declaration: `function AddObject(const S: string;AObject: TObject) : Integer; Virtual`

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 of 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` (390), `TStrings.Strings` (403), `TStrings.Objects` (402), `TStrings.InsertObject` (395)

2.71.7 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` (390), only it does not return the index of the inserted string.

See also: `TStrings.Add` (390)

2.71.8 TStrings.AddStrings

Synopsis: Add contents of another stringlist to this list.

Declaration: `procedure AddStrings(TheStrings: TStrings); Virtual; Overload`
`procedure AddStrings(const TheStrings: Array of string); Virtual`
`; Overload`

Visibility: public

Description: `AddStrings` adds the contents of `TheStrings` to the stringlist. Any associated objects are added as well.

See also: `TStrings.Add` (390), `TStrings.Assign` (391)

2.71.9 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` (195) 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` (195)

2.71.10 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` (390), `TStrings.AddStrings` (391), `TPersistent.Assign` (351)

2.71.11 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` (393) should be called to mark the end of the operation. Descendent 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. Descendent classes should also call these methods when modifying the string list.

Remark: Always put the corresponding call to `TStrings.EndUpdate` (393) 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` (393)

2.71.12 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` (401) is zero.

Since it is an abstract method, `TStrings` itself does not implement `Clear`. Descendent classes such as `TStringList` (383) implement this method.

See also: `TStrings.Objects` (402), `TStrings.Strings` (403), `TStrings.Delete` (392), `TStrings.Count` (401)

2.71.13 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`. Descendent classes such as `TStringList` (383) implement this method.

Errors: If `Index` is not in the allowed range, an `EStringListError` (230) is raised.

See also: `TStrings.Insert` (395), `TStrings.Objects` (402), `TStrings.Strings` (403), `TStrings.Clear` (392)

2.71.14 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` (392).

`TStrings` implements the `IFPObserved` (231) interface: when `EndUpdate` is called, a `ooChanged` notification is sent to all observers.

See also: `TStrings.BeginUpdate` (392), `IFPObserved` (231)

2.71.15 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` (402), `TStrings.Strings` (403), `TStrings.Count` (401), `TStrings.Assign` (391)

2.71.16 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` (230) exception is raised.

See also: `TStrings.Move` (396), `TStrings.Strings` (403), `TStrings.Count` (401)

2.71.17 TStrings.GetEnumerator

Synopsis: Create an `IEnumerator` instance

Declaration: `function GetEnumerator : TStringsEnumerator`

Visibility: public

Description: `GetEnumerator` is the implementation of the `IEnumerable` (1343) interface for `TStrings`. It creates a `TStringsEnumerator` (404) instance and returns it's `IEnumerator` (1343) interface.

See also: `TStringsEnumerator` (404), `IEnumerator` (1343), `IEnumerable` (1343)

2.71.18 TStrings.GetText

Synopsis: Returns the contents as a `PChar`

Declaration: `function GetText : PChar; Virtual`

Visibility: public

Description: `GetText` allocates a memory buffer and compies 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.

2.71.19 TStrings.IndexOf

Synopsis: Find a string in the list and return its position.

Declaration: `function IndexOf(const S: string) : Integer; Virtual`

Visibility: public

Description: `IndexOf` searches the list for `S`. The search is case-insensitive. 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` (395), `TStrings.IndexOfName` (394), `TStrings.Strings` (403)

2.71.20 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` (394), `TStrings.IndexOfObject` (395), `TStrings.Strings` (403)

2.71.21 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.

2.71.22 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` (395) instead.

Errors: If `Index` is less than zero or larger than `Count` then an `EStringListError` (230) exception is raised.

See also: `TStrings.Add` (390), `TStrings.InsertObject` (395), `TStrings.Append` (391), `TStrings.Delete` (392)

2.71.23 TStrings.InsertObject

Synopsis: Insert a string and associated object in the list.

Declaration: `procedure InsertObject(Index: Integer; const S: string; AObject: TObject)`

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` (230) exception is raised.

See also: `TStrings.Insert` (395), `TStrings.AddObject` (390), `TStrings.Append` (391), `TStrings.Delete` (392)

2.71.24 TStrings.LoadFromFile

Synopsis: Load the contents of a file as a series of strings.

Declaration: `procedure LoadFromFile(const FileName: string); Virtual`

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 (310) with the given filename, and then executes `TStrings.LoadfromStream` (396); after that the file stream object is destroyed again.

See also: `TStrings.LoadFromStream` (396), `TStrings.SaveToFile` (397), `Tstrings.SaveToStream` (397)

2.71.25 TStrings.LoadFromStream

Synopsis: Load the contents of a stream as a series of strings.

Declaration: `procedure LoadFromStream(Stream: TStream); Virtual`

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` (396), `TStrings.SaveToFile` (397), `Tstrings.SaveToStream` (397)

2.71.26 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` (230) is raised.

See also: `TStrings.Exchange` (393)

2.71.27 TStrings.SaveToFile

Synopsis: Save the contents of the list to a file.

Declaration: `procedure SaveToFile(const FileName: string); Virtual`

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 (310) with name `FileName`, calls `TStrings.SaveToStream` (397) and then destroys the file stream object.

Errors: An `EStreamError` (229) exception can be raised if the file `FileName` cannot be opened, or if it cannot be written to.

See also: `TStrings.SaveToStream` (397), `Tstrings.LoadFromStream` (396), `TStrings.LoadFromFile` (396)

2.71.28 TStrings.SaveToStream

Synopsis: Save the contents of the string to a stream.

Declaration: `procedure SaveToStream(Stream: TStream); Virtual`

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` (229) exception can be raised if the stream cannot be written to.

See also: `TStrings.SaveToFile` (397), `Tstrings.LoadFromStream` (396), `TStrings.LoadFromFile` (396)

2.71.29 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` (403)

2.71.30 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.

2.71.31 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` (400) 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` (400)

2.71.32 TStrings.TextLineBreakStyle

Synopsis: Determines which line breaks to use in the `Text` (403) 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:

tlbsLFLines are separated with a linefeed character #10.

tlbsCRLFLines are separated with a carriage-return/linefeed character pair: #13#10.

tlbsCRLines are separated with a carriage-return character #13.

It has no effect when setting the text property.

See also: `Text` (403)

2.71.33 TStrings.Delimiter

Synopsis: Delimiter character used in `DelimitedText` (399).

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` (399) property.

See also: `TStrings.DelimitedText` (399)

2.71.34 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` (399) and separated by the `Delimiter` (398) character.

Strings are quoted if they contain a space or any character with ASCII value less than 32.

The `CommaText` (400) 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` (399) 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.

See also: `TStrings.Delimiter` (398), `TStrings.Text` (403), `TStrings.QuoteChar` (399), `TStrings.CommaText` (400)

2.71.35 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` (399): under normal circumstances, quotes and spaces are considered specially (see the `TStrings.CommaText` (400) property for more information).

When `StrictDelimiter` is set to `True` then only the `Delimiter` (398) 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` (399), `CommaText` (400), `Delimiter` (398)

2.71.36 TStrings.QuoteChar

Synopsis: Quote character used in `DelimitedText` (399).

Declaration: `Property QuoteChar : Char`

Visibility: `public`

Access: `Read,Write`

Description: `QuoteChar` is the character used by the `DelimitedText` (399) property to quote strings that have a space or non-printing character in it.

2.71.37 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.

2.71.38 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` (400) character, or all characters if the `NameValueSeparator` character is not present.

2.71.39 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` (401) when read. Trying to set the capacity has no effect. Descendent classes such as `TStringList` (383) can override this property such that it actually sets the new capacity.

See also: `TStringList` (383), `TStrings.Count` (401)

2.71.40 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
NormalSttring
```

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. Spaces that occur next to a delimiter will be ignored. The following string:

```
"Comma,string" , "Quote""String",Space string,, NormalString
```

Will be converted to

```
Comma,String
Quote"String
Space
String

NormalString
```

This is a special case of the `TStrings.DelimitedText` (399) property where the quote character is always the double quote, and the delimiter is always the colon.

See also: `TStrings.Text` (403), `TStrings.SetText` (397)

2.71.41 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; descendent 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` (403), `TStrings.Objects` (402), `TStrings.Capacity` (400)

2.71.42 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` (402), `TStrings.IndexOfName` (394)

2.71.43 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 descendent classes to provide storage for the associated objects.

Errors: If an `Index` outside the valid range is specified, an `EStringListError` (230) exception will be raised.

See also: `TStrings.Strings` (403), `TStrings.IndexOfObject` (395), `TStrings.Names` (402), `TStrings.Values` (402)

2.71.44 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, its 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` (402), `TStrings.Strings` (403), `TStrings.Objects` (402)

2.71.45 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` (402) property. The name parts of name-value pairs can be accessed with the `TStrings.Names` (402) property, and the values can be set or read through the `TStrings.Values` (402) property.

Searching through the list can be done using the `TStrings.IndexOf` (394) method.

Errors: If `Index` is outside the allowed range, an `EStringListError` (230) exception is raised.

See also: `TStrings.Count` (401), `TStrings.Objects` (402), `TStrings.Names` (402), `TStrings.Values` (402), `TStrings.IndexOf` (394)

2.71.46 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` (400) property instead.

See also: `TStrings.Strings` (403), `TStrings.Count` (401), `TStrings.CommaText` (400)

2.71.47 TStrings.StringsAdapter

Synopsis: Not implemented in Free Pascal.

Declaration: `Property StringsAdapter : IStringsAdapter`

Visibility: `public`

Access: `Read, Write`

Description: Not implemented in Free Pascal.

2.72 TStringsEnumerator

2.72.1 Description

`TStringsEnumerator` implements the `#rtl.system.IEnumerator` (1343) interface for the `TStrings` (388) class, so the `TStrings` class can be used in a `for ... in` loop. It is returned by the `TStrings.GetEnumerator` (394) method of `TStrings`.

See also: `TStrings` (388), `TStrings.GetEnumerator` (394), `#rtl.system.IEnumerator` (1343)

2.72.2 Method overview

Page	Property	Description
404	<code>Create</code>	Initialize a new instance of <code>TStringsEnumerator</code>
404	<code>GetCurrent</code>	Return the current pointer in the list
405	<code>MoveNext</code>	Move the position of the enumerator to the next position in the list.

2.72.3 Property overview

Page	Property	Access	Description
405	<code>Current</code>	<code>r</code>	Current pointer in the list

2.72.4 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` (388)

2.72.5 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` (405), `TStringItem` (209)

2.72.6 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` (404)

2.72.7 TStringsEnumerator.Current

Synopsis: Current pointer in the list

Declaration: `Property Current : string`

Visibility: public

Access: Read

Description: `Current` redefines `GetCurrent` (404) as a property.

See also: `GetCurrent` (404)

2.73 TStringStream

2.73.1 Description

`TStringStream` stores its data in an `ansistring`. The contents of this string is available as the `DataStream` (407) property. It also introduces some methods to read or write parts of the stream'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` (369), `TStringStream.DataStream` (407), `TStringStream.ReadString` (406), `TStringStream.WriteString` (407)

2.73.2 Method overview

Page	Property	Description
406	Create	Creates a new stringstream and sets its initial content.
406	Read	Reads from the stream.
406	ReadString	Reads a string of length <code>Count</code>
406	Seek	Sets the position in the stream.
407	Write	<code>Write</code> overrides the <code>TStream.Write</code> (370) method.
407	WriteString	<code>WriteString</code> writes a string to the stream.

2.73.3 Property overview

Page	Property	Access	Description
407	<code>DataStream</code>	<code>r</code>	Contains the contents of the stream in string form

2.73.4 TStringStream.Create

Synopsis: Creates a new stringstream and sets its initial content.

Declaration: `constructor Create(const AString: string)`

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.

See also: `TStringStream.DataString` ([407](#))

2.73.5 TStringStream.Read

Synopsis: Reads from the stream.

Declaration: `function Read(var Buffer; Count: LongInt) : LongInt; Override`

Visibility: `public`

Description: `Read` overrides the `Read` ([370](#)) from `TStream` ([369](#)). It tries to read `Count` bytes into `Buffer`. It returns the number of bytes actually read. The position of the stream is advanced with the number of bytes actually read; When the reading has reached the end of the `DataStream` ([407](#)), then the reading stops, i.e. it is not possible to read beyond the end of the `datastring`.

See also: `TStream.Read` ([370](#)), `TStringStream.Write` ([407](#)), `TStringStream.DataString` ([407](#))

2.73.6 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` ([406](#)) method. It does not do the same string as the `TStream.ReadAnsiString` ([376](#)) method, which first reads a length integer to determine the length of the string to be read.

See also: `TStringStream.Read` ([406](#)), `TStream.ReadAnsiString` ([376](#))

2.73.7 TStringStream.Seek

Synopsis: Sets the position in the stream.

Declaration: `function Seek(Offset: LongInt; Origin: Word) : LongInt; Override`

Visibility: `public`

Description: `Seek` implements the abstract `Seek` ([371](#)) method.

2.73.8 TStringStream.Write

Synopsis: `Write` overrides the `TStream.Write` (370) method.

Declaration: `function Write(const Buffer; Count: LongInt) : LongInt; Override`

Visibility: `public`

Description: `Write` overrides the `TStream.Write` (370) method.

2.73.9 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.

2.73.10 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

2.74 TTextObjectWriter

2.74.1 Description

Not yet implemented.

2.75 TThread

2.75.1 Description

The `TThread` class encapsulates the native thread support of the operating system. To create a thread, declare a descendent of the `TThread` object and override the `Execute` (408) method. In this method, the `tthread`'s code should be executed. To run a thread, create an instance of the `tthread` descendent, and call its `execute` method.

See also: `EThread` (230), `TThread.Execute` (408)

2.75.2 Method overview

Page	Property	Description
412	AfterConstruction	Code to be executed after construction but before execute.
410	CheckTerminated	Check if the current thread has finished executing.
409	Create	Creates a new thread.
410	CreateAnonymousThread	Execute code in an anonymous thread
409	Destroy	Destroys the thread object.
408	Execute	Execute method. Must be overridden in a descendent thread.
412	GetSystemTimes	Return CPU stats
412	GetTickCount	Return tick count (32-bit)
412	GetTickCount64	Return tick count (64-bit)
410	NameThreadForDebugging	Set a thread name
409	Queue	Queue a method for execution in the main thread
411	RemoveQueuedEvents	Remove methods scheduled for execution from queue
413	Resume	Resumes the thread's execution. Deprecated, see <code>TThread.Start</code>
410	SetReturnValue	Set return value of a thread
411	Sleep	Prevent thread execution
411	SpinWait	Prevent thread execution in a spin-wait loop
413	Start	Starts a thread that was created in a suspended state.
413	Suspend	Suspends the thread's execution.
409	Synchronize	Synchronizes the thread by executing the method in the main thread.
413	Terminate	Signals the thread it should terminate.
413	WaitFor	Waits for the thread to terminate and returns the exit status.
412	Yield	Yield execution to other threads

2.75.3 Property overview

Page	Property	Access	Description
414	CurrentThread	r	Return current thread instance
415	ExternalThread	r	Is the thread instance an external thread ?
416	FatalException	r	Exception that occurred during thread execution
416	Finished	r	Has the thread finished executing
415	FreeOnTerminate	rw	Indicates whether the thread should free itself when it stops executing.
415	Handle	r	Returns the thread handle.
414	IsSingleProcessor	r	Is the current system single processor or not
416	OnTerminate	rw	Event called when the thread terminates.
415	Priority	rw	Returns the thread priority.
414	ProcessorCount	r	Return the processor count for this system
415	Suspended	rw	Indicates whether the thread is suspended.
416	ThreadID	r	Returns the thread ID.

2.75.4 TThread.Execute

Synopsis: Execute method. Must be overridden in a descendent thread.

Declaration: `procedure Execute; Virtual; Abstract`

Visibility: `protected`

Description: `Execute` is a method that must be overridden in descendent classes of the thread. It must contain the code that must execute in the thread. The `Execute` method is responsible for checking `Terminated` (195) at regular intervals: when it is set to `True` the execute method must exit.

See also: `Terminated` (195)

2.75.5 `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.

2.75.6 `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` (409), `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.

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 finishes it's execution, all its queued calls are removed from the queue list.

See also: `TThread.Synchronize` (409), `TThread.RemoveQueuedEvents` (411)

2.75.7 `TThread.Create`

Synopsis: Creates a new thread.

Declaration: `constructor Create (CreateSuspended: Boolean; const StackSize: SizeUInt)`

Visibility: `public`

Description: Creates a new thread.

2.75.8 `TThread.Destroy`

Synopsis: Destroys the thread object.

Declaration: `destructor Destroy; Override`

Visibility: `public`

Description: Destroys the thread object.

2.75.9 TThread.CreateAnonymousThread

Synopsis: Execute code in an anonymous thread

Declaration: `class function CreateAnonymousThread(aProc: TProcedure); Static`

Visibility: public

Description: `CreateAnonymousThread` will create an instance of a `TThread` descendent 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` ([410](#))

2.75.10 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 ansistring. This is mainly useful for debugging purposes, as thread names are more easily recognizable than IDs.

Note that this requires OS support. Currently this is not implemented for any FPC platform and is supported for Delphi compatibility only.

2.75.11 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` ([230](#)) exception is raised.

See also: `EThreadExternalException` ([230](#)), `TThread.CheckTerminated` ([410](#))

2.75.12 TThread.CheckTerminated

Synopsis: Check if the current thread has finished executing.

Declaration: `class function CheckTerminated; 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` (230) exception is raised.

See also: `EThreadExternalException` (230), `TThread.SetReturnValue` (410)

2.75.13 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. If both arguments are specified, then all calls to the given method by the given thread are removed.

See also: `TThread.Synchronize` (409), `TThread.Queue` (409)

2.75.14 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` (195) can cause a thread switch). The input parameter (alterations) specifies the number of spin loops.

See also: `SysUtils.Sleep` (195), `TThread.Sleep` (411)

2.75.15 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` (195)

In difference with `TThread.SpinWait` (411), a thread switch may occur during the sleep.

See also: `SysUtils.Sleep` (195), `TThread.SpinWait` (411)

2.75.16 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.

2.75.17 TThread.GetSystemTimes

Synopsis: Return CPU stats

Declaration: `class procedure GetSystemTimes(out aSystemTimes: TSystemTimes); Static`

Visibility: `public`

Description: `GetSystemTimes` is provided for Delphi compatibility only, it currently returns empty values only.

See also: `TSystemTimes` ([195](#))

2.75.18 TThread.GetTickCount

Synopsis: Return tick count (32-bit)

Declaration: `class function GetTickCount; Static`

Visibility: `public`

Description: `GetTickCount` is deprecated and should not be used. Use `TThread.GetTickCount64` ([412](#)) instead.

See also: `TThread.GetTickCount64` ([412](#))

2.75.19 TThread.GetTickCount64

Synopsis: Return tick count (64-bit)

Declaration: `class function GetTickCount64; Static`

Visibility: `public`

Description: `GetTickCount64` simply calls `SysUtils.GetTickCount64` ([195](#)) and is implemented for Delphi compatibility only.

See also: `SysUtils.GetTickCount64` ([195](#))

2.75.20 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 starts the thread if it was created in a suspended state. When overriding this method, the inherited method must be called.

2.75.21 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` ([409](#))

2.75.22 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` ([413](#)), `TThread.Suspend` ([413](#))

2.75.23 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` ([413](#)), `TThread.Start` ([413](#))

2.75.24 TThread.Terminate

Synopsis: Signals the thread it should terminate.

Declaration: `procedure Terminate`

Visibility: `public`

Description: Signals the thread it should terminate.

2.75.25 TThread.WaitFor

Synopsis: Waits for the thread to terminate and returns the exit status.

Declaration: `function WaitFor : Integer`

Visibility: `public`

Description: Waits for the thread to terminate and returns the exit status.

2.75.26 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` ([415](#))

2.75.27 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` ([414](#))

2.75.28 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` ([414](#)) is less than or equal to 1, `False` otherwise.

See also: `TThread.ProcessorCount` ([414](#))

2.75.29 TThread.FreeOnTerminate

Synopsis: Indicates whether the thread should free itself when it stops executing.

Declaration: `Property FreeOnTerminate : Boolean`

Visibility: `public`

Access: `Read,Write`

Description: Indicates whether the thread should free itself when it stops executing.

2.75.30 TThread.Handle

Synopsis: Returns the thread handle.

Declaration: `Property Handle : TThreadID`

Visibility: `public`

Access: `Read`

Description: Returns the thread handle.

2.75.31 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` ([414](#)).

See also: `TThread.CurrentThread` ([414](#))

2.75.32 TThread.Priority

Synopsis: Returns the thread priority.

Declaration: `Property Priority : TThreadPriority`

Visibility: `public`

Access: `Read,Write`

Description: Returns the thread priority.

2.75.33 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.

2.75.34 TThread.Finished

Synopsis: Has the thread finished executing

Declaration: `Property Finished : Boolean`

Visibility: `public`

Access: `Read`

Description: `Finished` is `True` when `TThread.Execue` (407) has finished executing, but the thread is still cleaning up (calling `OnTerminate`, etc).

See also: `TThread.Execue` (407), `TThread.OnTerminate` (416)

2.75.35 TThread.ThreadID

Synopsis: Returns the thread ID.

Declaration: `Property ThreadID : TThreadID`

Visibility: `public`

Access: `Read`

Description: Returns the thread ID.

2.75.36 TThread.OnTerminate

Synopsis: Event called when the thread terminates.

Declaration: `Property OnTerminate : TNotifyEvent`

Visibility: `public`

Access: `Read,Write`

Description: Event called when the thread terminates.

2.75.37 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.

2.76 TThreadList

2.76.1 Description

`TThreadList` is a thread-safe `TList` (330) 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` (330)

2.76.2 Method overview

Page	Property	Description
417	Add	Adds an element to the list.
418	Clear	Removes all emements from the list.
417	Create	Creates a new thread-safe list.
417	Destroy	Destroys the list instance.
418	LockList	Locks the list for exclusive access.
418	Remove	Removes an item from the list.
418	UnlockList	Unlocks the list after it was locked.

2.76.3 Property overview

Page	Property	Access	Description
419	Duplicates	rw	Describes what to do with duplicates

2.76.4 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` ([417](#))

2.76.5 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` ([417](#))

2.76.6 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` ([418](#)), `Clear` ([418](#)), `Remove` ([418](#)), `UnlockList` ([418](#))

2.76.7 TThreadList.Clear

Synopsis: Removes all emements 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` (418), `Add` (417), `Remove` (418), `UnlockList` (418)

2.76.8 TThreadList.LockList

Synopsis: Locks the list for exclusive access.

Declaration: `function LockList : TList`

Visibility: `public`

Description: `LockList` locks the list for exclusive access. Locklist uses an internal critical section, so all rules for multiple locking of critical sections apply to locklist/unlocklist as well.

See also: `Clear` (418), `Add` (417), `Remove` (418), `UnlockList` (418)

2.76.9 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` (418), `Add` (417), `Clear` (418), `UnlockList` (418)

2.76.10 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` (418), `Add` (417), `Remove` (418), `lockList` (418)

2.76.11 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` (386) property of `TStringList` (383).

See also: `TDuplicates` (201)

2.77 TWriter

2.77.1 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` (250) 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` (373) call should be used.

See also: `TFile` (308), `TWriter` (419), `TAbstractObjectReader` (243)

2.77.2 Method overview

Page	Property	Description
420	Create	Creates a new Writer with a stream and bufsize.
421	DefineBinaryProperty	Callback used when defining and streaming custom properties.
421	DefineProperty	Callback used when defining and streaming custom properties.
421	Destroy	Destroys the writer instance.
421	Write	Write raw data to stream
421	WriteBoolean	Write boolean value to the stream.
422	WriteChar	Write a character to the stream.
422	WriteCollection	Write a collection to the stream.
422	WriteComponent	Stream a component to the stream.
423	WriteCurrency	Write a currency value to the stream
423	WriteDate	Write a date to the stream.
422	WriteDescendent	Write a descendent component to the stream.
422	WriteFloat	Write a float to the stream.
423	WriteIdent	Write an identifier to the stream.
423	WriteInteger	Write an integer to the stream.
424	WriteListBegin	Write a start-of-list marker to the stream.
424	WriteListEnd	Write an end-of-list marker to the stream.
424	WriteRootComponent	Write a root component to the stream.
424	WriteSet	Write a set value to the stream
423	WriteSingle	Write a single-type real to the stream.
424	WriteString	Write a string to the stream.
425	WriteUnicodeString	Write a unicode string to the stream.
425	WriteVariant	Write a variant to the stream
422	WriteWideChar	Write widechar to stream
425	WriteWideString	Write a widestring value to the stream

2.77.3 Property overview

Page	Property	Access	Description
426	Driver	r	Driver used when writing to the stream.
425	OnFindAncestor	rw	Event occurring when an ancestor component must be found.
426	OnWriteMethodProperty	rw	Handler from writing method properties.
426	OnWriteStringProperty	rw	Event handler for translating strings written to stream.
426	PropertyPath	r	Path to the property that is currently being written
425	RootAncestor	rw	Ancestor of root component.

2.77.4 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.

2.77.5 TWriter.Destroy

Synopsis: Destroys the writer instance.

Declaration: `destructor Destroy; Override`

Visibility: `public`

Description: Destroys the writer instance.

2.77.6 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.

2.77.7 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.

2.77.8 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` ([270](#)), `TAbstractObjectWriter.Write` ([252](#))

2.77.9 TWriter.WriteBoolean

Synopsis: Write boolean value to the stream.

Declaration: `procedure WriteBoolean(Value: Boolean)`

Visibility: `public`

Description: Write boolean value to the stream.

2.77.10 TWriter.WriteCollection

Synopsis: Write a collection to the stream.

Declaration: `procedure WriteCollection(Value: TCollection)`

Visibility: `public`

Description: Write a collection to the stream.

2.77.11 TWriter.WriteComponent

Synopsis: Stream a component to the stream.

Declaration: `procedure WriteComponent(Component: TComponent)`

Visibility: `public`

Description: Stream a component to the stream.

2.77.12 TWriter.WriteChar

Synopsis: Write a character to the stream.

Declaration: `procedure WriteChar(Value: Char)`

Visibility: `public`

Description: Write a character to the stream.

2.77.13 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` ([359](#)), `TWriter.WriteWideString` ([425](#))

2.77.14 TWriter.WriteDescendent

Synopsis: Write a descendent component to the stream.

Declaration: `procedure WriteDescendent(ARoot: TComponent; AAncestor: TComponent)`

Visibility: `public`

Description: Write a descendent component to the stream.

2.77.15 TWriter.WriteFloat

Synopsis: Write a float to the stream.

Declaration: `procedure WriteFloat(const Value: Extended)`

Visibility: `public`

Description: Write a float to the stream.

2.77.16 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.

2.77.17 TWriter.WriteDate

Synopsis: Write a date to the stream.

Declaration: `procedure WriteDate(const Value: TDateTime)`

Visibility: `public`

Description: Write a date to the stream.

2.77.18 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` ([360](#))

2.77.19 TWriter.WriteIdent

Synopsis: Write an identifier to the stream.

Declaration: `procedure WriteIdent(const Ident: string)`

Visibility: `public`

Description: Write an identifier to the stream.

2.77.20 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.

2.77.21 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` ([361](#))

2.77.22 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.

2.77.23 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.

2.77.24 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.

2.77.25 TWriter.WriteString

Synopsis: Write a string to the stream.

Declaration: `procedure WriteString (const Value: string)`

Visibility: public

Description: Write a string to the stream.

2.77.26 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` ([362](#))

2.77.27 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.WriteUnicodeString` ([272](#)), `TReader.ReadUnicodeString` ([362](#))

2.77.28 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` ([272](#)), `TReader.ReadVariant` ([362](#))

2.77.29 TWriter.RootAncestor

Synopsis: Ancestor of root component.

Declaration: `Property RootAncestor : TComponent`

Visibility: public

Access: Read, Write

Description: Ancestor of root component.

2.77.30 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.

2.77.31 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` (210) for a description of the arguments.

See also: `TWriteMethodPropertyEvent` (210), `TReader.OnSetMethodProperty` (364)

2.77.32 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` (207) for a description of the various parameters.

See also: `TReader.OnPropertyNotFound` (364), `TReader.OnSetMethodProperty` (364), `TReadWriteStringPropertyEvent` (207)

2.77.33 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.

2.77.34 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` (350) descendent class properties are written.

Chapter 3

Reference for unit 'clocale'

3.1 Overview

The `clocale` offers no API by itself: it just initializes the internationalization settings of the `sysutils` (1360) 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 4

Reference for unit 'cmem'

4.1 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` ([429](#)), `Free` ([428](#)) and `ReAlloc` ([429](#)). 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.

4.2 Constants, types and variables

4.2.1 Constants

`LibName = 'c'`

`LibName` is the name of the library that is actually used. On most systems, this is simply "libc.so".

4.3 Procedures and functions

4.3.1 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` ([429](#)), `Free` ([428](#)), `Realloc` ([429](#))

4.3.2 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` ([429](#)), `ReAlloc` ([429](#))

4.3.3 Malloc

Synopsis: `Malloc` external declaration.

Declaration: `function Malloc(Size: ptruint) : Pointer`

Visibility: `default`

Description: `Malloc` is the external declaration of the C library's `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` ([428](#)), `ReAlloc` ([429](#))

4.3.4 ReAlloc

Synopsis: `ReAlloc` re-allocates 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` ([429](#)), `Free` ([428](#))

Chapter 5

Reference for unit 'Crt'

5.1 Overview

This chapter describes the CRT unit for Free Pascal, both under 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 correctly work.

5.2 Constants, types and variables

5.2.1 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

5.2.2 Types

`PConsoleBuf = ^TConsoleBuf`

```
TCharAttr = packed record
  ch : Char;
  attr : Byte;
end
```

`TConsoleBuf = Array[0..ConsoleMaxX*ConsoleMaxY-1] of TCharAttr`

`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.

5.2.3 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

5.3 Procedures and functions

5.3.1 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` ([445](#))

Listing: `./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.

```

5.3.2 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` ([437](#)), `InsLine` ([439](#)), `ClrScr` ([435](#))

Listing: `./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.
```

5.3.3 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` ([445](#))

Listing: `./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.
```

5.3.4 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` ([436](#)), `CursorOff` ([436](#))

5.3.5 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` ([436](#)), `CursorBig` ([436](#))

5.3.6 cursoron

Synopsis: Display cursor

Declaration: `procedure cursoron`

Visibility: default

Description: `CursorOn` switches the cursor on. Not implemented on unixes.

Errors: None.

See also: `CursorBig` ([436](#)), `CursorOff` ([436](#))

5.3.7 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` ([442](#)), `NoSound` ([441](#))

Listing: `./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.
```

5.3.8 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` ([435](#)), `InsLine` ([439](#)), `ClrScr` ([435](#))

Listing: `./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.

```

5.3.9 GotoXY

Synopsis: Set cursor position on screen.

Declaration: `procedure GotoXY(X: tcrtcoord; Y: tcrtcoord)`

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 \(444\)](#), [WhereY \(444\)](#), [Window \(445\)](#)

Listing: `./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.

```

5.3.10 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 \(443\)](#), [TextBackground \(442\)](#), [LowVideo \(440\)](#), [NormVideo \(440\)](#)

Listing: ./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.

```

5.3.11 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` ([435](#)), `DelLine` ([437](#)), `ClrScr` ([435](#))

Listing: ./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 position' );
  GotoXY(1,3);
  ReadKey;
  InsLine;
  Write( 'Line 2 ' );
  GotoXY(1,10);
end.

```

5.3.12 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` ([441](#))

Listing: `./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.
```

5.3.13 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` ([438](#))

Errors: None.

See also: `TextColor` ([443](#)), `TextBackground` ([442](#)), `HighVideo` ([438](#)), `NormVideo` ([440](#))

5.3.14 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` ([438](#))

Errors: None.

See also: `TextColor` ([443](#)), `TextBackground` ([442](#)), `LowVideo` ([440](#)), `HighVideo` ([438](#))

5.3.15 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` ([442](#))

Listing: `./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');
  while (i < 15000) do
    begin
      inc(i, 500);
      Sound(i);
      Delay(100);
    end;
  WriteLn('Quiet now!');
  NoSound; { Stop noise }
end.
```

5.3.16 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` ([439](#))

Listing: `./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.

```

5.3.17 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 ([441](#))

5.3.18 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 ([443](#)), HighVideo ([438](#)), LowVideo ([440](#)), NormVideo ([440](#))

Listing: ./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.

```

5.3.19 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` ([442](#)), `HighVideo` ([438](#)), `LowVideo` ([440](#)), `NormVideo` ([440](#))

Listing: `./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.

```

5.3.20 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` ([445](#))

5.3.21 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` (438), `WhereY` (444), `Window` (445)

Listing: `./crtex/ex7.pp`

```
Program Example7;  
uses Crt;  
  
{ Program to demonstrate the WhereX and WhereY functions. }  
  
begin  
  WriteLn( 'Cursor postion: X=',WhereX, ' Y=',WhereY );  
end.
```

5.3.22 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` (438), `WhereX` (444), `Window` (445)

Listing: `./crtex/ex7.pp`

```
Program Example7;  
uses Crt;  
  
{ Program to demonstrate the WhereX and WhereY functions. }  
  
begin  
  WriteLn( 'Cursor postion: X=',WhereX, ' Y=',WhereY );  
end.
```

5.3.23 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` ([438](#)), `WhereX` ([444](#)), `WhereY` ([444](#)), `ClrScr` ([435](#))

Listing: `./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.

```

Chapter 6

Reference for unit 'cthreads'

6.1 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` ([1118](#)) unit. This is done using the `SetCThreadManager` ([446](#)) 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.

6.2 Procedures and functions

6.2.1 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 7

Reference for unit 'ctypes'

7.1 Used units

Table 7.1: Used units by unit 'ctypes'

Name	Page
System	1118
unixtype	1623

7.2 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.

7.3 Constants, types and variables

7.3.1 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 = packed Array[0..15] of Byte`

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` (447) type.

`pcchar = UnixType.pcchar`

Pointer to `cchar` (447) type.

`pcdouble = UnixType.pcdouble`

Pointer to `cdouble` (447) type.

`pcfloat = UnixType.pcfloating`

Pointer to `cfloat` (448) type.

`pcint = UnixType.pcint`

Pointer to `cint` (448) type.

`pcint16 = UnixType.pcint16`

Pointer to `cint16` (448) type.

`pcint32 = UnixType.pcint32`

Pointer to `cint32` (448) type.

`pcint64 = UnixType.pcint64`

Pointer to `cint64` (448) type.

`pcint8 = UnixType.pcint8`

Pointer to `cint8` (448) type.

`pclong = UnixType.pclong`

Pointer to `clong` (448) type.

`pclongdouble = ^clongdouble`

Pointer to `clongdouble` (448) type.

`pclonglong = UnixType.pclonglong`

Pointer to `clonglong` (448) type.

`pcschar = UnixType.pcschar`

Pointer to `cschar` (448) type.

`pcshort = UnixType.pcshort`

Pointer to `cshort` (448) type.

`pcsigned = UnixType.pcsigned`

Pointer to `csigned` (448) type.

`pcsint = UnixType.pcsint`

Pointer to `csint` (449) type.

`pcsize_t = UnixType.psize_t`

Pointer to generic size type

`pcslong = UnixType.pcslong`

Pointer to `clong` (449) type.

`pcslonglong = UnixType.pcslonglong`

Pointer to `cslonglong` (449) type.

`pcsshort = UnixType.pcsshort`

Pointer to `csshort` (449) type.

`pcuchar = UnixType.pcuchar`

Pointer to `cuchar` (449) type.

`pcuint = UnixType.pcuint`

Pointer to `cuint` (449) type.

`pcuint16 = UnixType.pcuint16`

Pointer to `cuint16` (449) type.

`pcuint32 = UnixType.pcuint32`

Pointer to `cuint32` (449) type.


```
pcuint64 = UnixType.pcuint64
```

Pointer to `cuint64` (449) type.

```
pcuint8 = UnixType.pcuint8
```

Pointer to `cuint8` (449) type.

```
pculong = UnixType.pculong
```

Pointer to `culong` (449) type.

```
pculonglong = UnixType.pculonglong
```

Pointer to `culonglong` (449) type.

```
pcunsigned = UnixType.punsigned
```

Pointer to `cunsigned` (450) type.

```
pcushort = UnixType.pcushort
```

Pointer to `cushort` (450) type.

7.4 Procedures and functions

7.4.1 operator `*(clongdouble, Double): Double`

Synopsis: Implement multiplication of `clongdouble` and `double`.

Declaration:

```
operator operator *(clongdouble, Double): Double(const c: clongdouble;
                                                    const e: Double)
                : Double
```

Visibility: default

Description: This operator allows to multiply a `double` typed value with a `clongdouble` typed value. the result is a `double` typed value.

7.4.2 operator `*(Double, clongdouble): Double`

Synopsis: Implement multiplication of `double` and `clongdouble`.

Declaration:

```
operator operator *(Double, clongdouble): Double(const e: Double;
                                                    const c: clongdouble)
                : Double
```

Visibility: default

Description: This operator allows to multiply a `clongdouble` typed value with a `double` typed value. The result is a `double` typed value.

7.4.3 operator +(clongdouble, Double): Double

Synopsis:

Declaration: `operator operator +(clongdouble, Double): Double(const c: clongdouble;
const e: Double)
: Double`

Visibility: default

Description: This operator allows to add `clongdouble` and `double` typed values. The result is a `double` typed value.

7.4.4 operator +(Double, clongdouble): Double

Synopsis: Implement addition of `clongdouble` and `double`.

Declaration: `operator operator +(Double, clongdouble): Double(const e: Double;
const c: clongdouble)
: Double`

Visibility: default

Description: This operator allows to add `double` and `clongdouble` typed values. The result is a `double` typed value.

7.4.5 operator -(clongdouble, Double): Double

Synopsis: Implement subtraction of `double` and `clongdouble`.

Declaration: `operator operator -(clongdouble, Double): Double(const c: clongdouble;
const e: Double)
: Double`

Visibility: default

Description: This operator allows to subtract a `double` typed value from a `clongdouble` typed value. The result is a `double` typed value.

7.4.6 operator -(Double, clongdouble): Double

Synopsis: Implement subtraction of `clongdouble` and `double`.

Declaration: `operator operator -(Double, clongdouble): Double(const e: Double;
const c: clongdouble)
: Double`

Visibility: default

Description: This operator allows to subtract a `clongdouble` typed value from a `double` typed value. The result is a `double` typed value.

7.4.7 operator /(clongdouble, Double): Double

Synopsis: Implement division of double and clongdouble.

Declaration: `operator operator /(clongdouble, Double): Double(const c: clongdouble;
const e: Double)
: Double`

Visibility: default

Description: This operator allows to divide a clongdouble typed value by a double typed value. the result is a double typed value.

7.4.8 operator /(Double, clongdouble): Double

Synopsis: Implement division of clongdouble and double.

Declaration: `operator operator /(Double, clongdouble): Double(const e: Double;
const c: clongdouble)
: Double`

Visibility: default

Description: This operator allows to divide a double typed value by a clongdouble typed value. the result is a double typed value.

7.4.9 operator :=(clongdouble): Double

Synopsis: Implement assignment of a clongdouble to a double type.

Declaration: `operator operator :=(clongdouble): Double(const v: clongdouble) : Double`

Visibility: default

Description: This operator allows to assign a clongdouble to a Double type.

7.4.10 operator :=(Double): clongdouble

Synopsis: Implement assignment of a double to a clongdouble type.

Declaration: `operator operator :=(Double): clongdouble(const v: Double) : clongdouble`

Visibility: default

Description: This operator allows to assign a double to a clongdouble type.

7.4.11 operator <(clongdouble, Double): Boolean

Synopsis: Implement less than comparison between double and clongdouble.

Declaration: `operator operator <(clongdouble, Double): Boolean(const c: clongdouble;
const e: Double)
: Boolean`

Visibility: default

Description: This operator compares the values of a clongdouble typed value and a double typed value, and returns True if the clongdouble value is less than the double value.

7.4.12 operator <(Double, clongdouble): Boolean

Synopsis: Implement less than comparison between `clongdouble` and `double`.

Declaration:

```
operator operator <(Double, clongdouble): Boolean(const e: Double;
                                                    const c: clongdouble)
                                                    : Boolean
```

Visibility: default

Description: This operator compares the values of a `double` typed value and a `clongdouble` typed value, and returns `True` if the `double` value is less than the `clongdouble` value.

7.4.13 operator <=(clongdouble, Double): Boolean

Synopsis: Implement greater than or equal comparison between `double` and `clongdouble`.

Declaration:

```
operator operator <=(clongdouble, Double): Boolean(const c: clongdouble;
                                                    const e: Double)
                                                    : Boolean
```

Visibility: default

Description: This operator compares the values of a `clongdouble` typed value and a `double` typed value, and returns `True` if the `clongdouble` value is less than or equal to the `double` value.

7.4.14 operator <=(Double, clongdouble): Boolean

Synopsis: Implement less than or equal comparison between `clongdouble` and `double`.

Declaration:

```
operator operator <=(Double, clongdouble): Boolean(const e: Double;
                                                    const c: clongdouble)
                                                    : Boolean
```

Visibility: default

Description: This operator compares the values of a `double` typed value and a `clongdouble` typed value, and returns `True` if the `double` value is less than or equal to the `clongdouble` value.

7.4.15 operator =(clongdouble, Double): Boolean

Synopsis: Implement equality of `clongdouble` and `double`.

Declaration:

```
operator operator =(clongdouble, Double): Boolean(const c: clongdouble;
                                                    const e: Double)
                                                    : Boolean
```

Visibility: default

Description: This operator compares the values of a `double` typed value and a `clongdouble` typed value, and returns `True` if the values are equal (only `double` precision is used).

7.4.16 operator ==(Double, clongdouble): Boolean

Synopsis: Implement equality of double and clongdouble.

Declaration: `operator operator ==(Double, clongdouble): Boolean(const e: Double;
const c: clongdouble)
: Boolean`

Visibility: default

Description: This operator compares the values of a `clongdouble` typed value and a `double` typed value, and returns `True` if the values are equal (only double precision is used).

7.4.17 operator >(clongdouble, Double): Boolean

Synopsis: Implement greater than comparison between double and clongdouble.

Declaration: `operator operator >(clongdouble, Double): Boolean(const c: clongdouble;
const e: Double)
: Boolean`

Visibility: default

Description: This operator compares the values of a `clongdouble` typed value and a `double` typed value, and returns `True` if the `clongdouble` value is greater than the `double` value.

7.4.18 operator >(Double, clongdouble): Boolean

Synopsis: Implement greater than comparison between clongdouble and double.

Declaration: `operator operator >(Double, clongdouble): Boolean(const e: Double;
const c: clongdouble)
: Boolean`

Visibility: default

Description: This operator compares the values of a `double` typed value and a `clongdouble` typed value, and returns `True` if the `double` value is greater than the `clongdouble` value.

7.4.19 operator >=(clongdouble, Double): Boolean

Synopsis: Implement greater than or equal comparison between double and clongdouble.

Declaration: `operator operator >=(clongdouble, Double): Boolean(const c: clongdouble;
const e: Double)
: Boolean`

Visibility: default

Description: This operator compares the values of a `clongdouble` typed value and a `double` typed value, and returns `True` if the `clongdouble` value is greater than or equal to the `double` value.

7.4.20 operator >=(Double, clongdouble): Boolean

Synopsis: Implement greater than or equal comparison between `clongdouble` and `double`.

Declaration:

```
operator operator >=(Double, clongdouble): Boolean(const e: Double;  
                                                    const c: clongdouble)  
                                                    : Boolean
```

Visibility: default

Description: This operator compares the values of a `double` typed value and a `clongdouble` typed value, and returns `True` if the `double` value is greater than or equal to the `clongdouble` value.

Chapter 8

Reference for unit 'cwstring'

8.1 Overview

The `cstring` unit offers no API by itself: it just initializes the widestring manager record of the system (1118) 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;
```

8.2 Procedures and functions

8.2.1 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 9

Reference for unit 'dateutils'

9.1 Used units

Table 9.1: Used units by unit 'dateutils'

Name	Page
math	774
System	1118
sysutils	1360

9.2 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.

9.3 Constants, types and variables

9.3.1 Constants

`ApproxDaysPerMonth : Double = 30.4375`

Average number of days in a month, measured over a year. Used in `MonthsBetween` ([506](#)).

`ApproxDaysPerYear : Double = 365.25`

Average number of days in a year, measured over 4 years. Used in `YearsBetween` ([548](#)).

`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

MonthsPerYear = 12

Number of months in a year

OneHour = 1 / HoursPerDay

One hour as a fraction of a day (suitable for TDateTime)

OneMillisecond = 1 / MSecsPerDay

One millisecond as a fraction of a day (suitable for TDateTime)

OneMinute = 1 / MinsPerDay

One minute as a fraction of a day (suitable for TDateTime)

OneSecond = 1 / SecsPerDay

One second as a fraction of a day (suitable for TDateTime)

RecodeLeaveFieldAsIs = (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 millenium

9.4 Procedures and functions

9.4.1 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` ([463](#)), `CompareDateTime` ([462](#)), `SameDate` ([516](#)), `SameTime` ([518](#)), `SameDateTime` ([517](#))

Listing: `./datutex/ex99.pp`

Program `Example99;`

`{ This program demonstrates the CompareDate function }`

Uses `SysUtils, DateUtils;`

Const

`Fmt = 'dddd dd mmm yyyy';`

Procedure `Test(D1,D2 : TDateTime);`

Var

`Cmp : Integer;`

```

Procedure Test(D1,D2 : TDateTime);

Var
  Cmp : Integer;

begin
  Write(FormatDateTime(Fmt,D1), ' is ');
  Cmp:=CompareDateTime(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.

```

9.4.3 CompareTime

Synopsis: Compares two times of the day, disregarding the date part.

Declaration: `function CompareTime(const A: TDateTime;const B: TDateTime)
: TValueRelationship`

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` (462), `CompareDate` (461), `SameDate` (516), `SameTime` (518), `SameDateTime` (517)

Listing: ./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 ');
    WriteIn(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.

```

9.4.4 DateOf

Synopsis: Extract the date part from a DateTime 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 Trunc.

See also: TimeOf (528), YearOf (547), MonthOf (506), DayOf (466), HourOf (481), MinuteOf (502), SecondOf (519), MilliSecondOf (497)

Listing: ./datutex/ex1.pp

Program Example1;

{ This program demonstrates the DateOf function }

Uses SysUtils, DateUtils;

Begin

 WriteLn('Date is: ', DateTimeToStr(DateOf(Now)));

End.

9.4.5 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:

0-4Seconds divided by 2

5-10Minutes

11-15Hours

16-20Day

21-24Month

25-31Years since 1980

See also: DosDateTimeToDateTime ([474](#))

9.4.6 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 ([496](#)), TryJulianDateToDateTime ([533](#)), DateTimeToModifiedJulianDate ([466](#)), TryModifiedJulianDateToDateTime ([533](#))

9.4.7 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 \(534\)](#), [MacToDateTime \(497\)](#), [MacTimeStampToUnix \(497\)](#)

9.4.8 DateTimeToModifiedJulianDate

Synopsis: Convert a `TDateTime` value to a modified Julian date representation

Declaration: `function DateTimeToModifiedJulianDate(const AValue: TDateTime) : Double`

Visibility: default

Description: Not yet implemented.

Errors: Currently, trying to use this function will raise an exception.

See also: [DateTimeToJulianDate \(465\)](#), [JulianDateToDateTime \(496\)](#), [TryJulianDateToDateTime \(533\)](#), [TryModifiedJulianDateToDateTime \(533\)](#)

9.4.9 DateTimeToUnix

Synopsis: Convert a `TDateTime` value to Unix epoch time

Declaration: `function DateTimeToUnix(const AValue: TDateTime) : 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 \(535\)](#)

9.4.10 DayOf

Synopsis: Extract the day (of month) part from a `DateTime` 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 \(547\)](#)

See also: [YearOf \(547\)](#), [WeekOf \(535\)](#), [MonthOf \(506\)](#), [HourOf \(481\)](#), [MinuteOf \(502\)](#), [SecondOf \(519\)](#), [MilliSecondOf \(497\)](#)

9.4.11 DayOfTheMonth

Synopsis: Extract the day (of month) part of a `DateTime` 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` (535) function.

See also: `DayOfTheYear` (467), `WeekOfTheMonth` (535), `HourOfTheMonth` (482), `MinuteOfTheMonth` (503), `SecondOfTheMonth` (520), `MilliSecondOfTheMonth` (499)

9.4.12 DayOfTheWeek

Synopsis: Extracts the day of the week from a `DateTime` 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` (467), `DayOfTheMonth` (466), `HourOfTheWeek` (482), `MinuteOfTheWeek` (503), `SecondOfTheWeek` (521), `MilliSecondOfTheWeek` (499)

Listing: `./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.

9.4.13 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` (536) function.

See also: `WeekOfTheYear` (536), `HourOfTheYear` (483), `MinuteOfTheYear` (503), `SecondOfTheYear` (521), `MilliSecondOfTheYear` (500)

9.4.14 DaysBetween

Synopsis: Number of whole days between two DateTime 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` (548), `MonthsBetween` (506), `WeeksBetween` (536), `HoursBetween` (483), `MinutesBetween` (504), `SecondsBetween` (521), `MillisecondsBetween` (500)

Listing: `./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.

9.4.15 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` (537), `WeeksInYear` (538), `DaysInYear` (470), `DaysInAYear` (469), `DaysInMonth` (470)

Listing: `./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.

9.4.16 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` (537), `WeeksInYear` (538), `DaysInYear` (470), `DaysInMonth` (470), `DaysInAMonth` (468)

Listing: `./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.

9.4.17 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` (537), `WeeksInYear` (538), `DaysInYear` (470), `DaysInAYear` (469), `DaysInAMonth` (468)

Listing: `./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.

9.4.18 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` (537), `WeeksInYear` (538), `DaysInAYear` (469), `DaysInMonth` (470), `DaysInAMonth` (468)

Listing: `./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.

9.4.19 DaySpan

Synopsis: Calculate the approximate number of days between two `DateTime` 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` (549), `MonthSpan` (507), `WeekSpan` (538), `HourSpan` (484), `MinuteSpan` (505), `SecondSpan` (522), `MilliSecondSpan` (501), `DaysBetween` (468)

Listing: `./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.

9.4.20 DecodeDateDay

Synopsis: Decode a `DateTime` 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` (475), `EncodeDateMonthWeek` (475), `EncodeDateWeek` (476), `EncodeDateDay` (475), `DecodeDateTime` (473), `DecodeDateWeek` (473), `DecodeDateMonthWeek` (472)

Listing: `./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.

9.4.21 DecodeDateMonthWeek

Synopsis: Decode a `DateTime` 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` (475), `EncodeDateMonthWeek` (475), `EncodeDateWeek` (476), `EncodeDateDay` (475), `DecodeDateTime` (473), `DecodeDateWeek` (473), `DecodeDateDay` (471)

Listing: `./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.

9.4.22 DecodeDateTime

Synopsis: Decode a datetime 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` (475), `EncodeDateMonthWeek` (475), `EncodeDateWeek` (476), `EncodeDateDay` (475), `DecodeDateWeek` (473), `DecodeDateDay` (471), `DecodeDateMonthWeek` (472)

Listing: `./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.

9.4.23 DecodeDateWeek

Synopsis: Decode a `DateTime` 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` (475), `EncodeDateMonthWeek` (475), `EncodeDateWeek` (476), `EncodeDateDay` (475), `DecodeDateTime` (473), `DecodeDateDay` (471), `DecodeDateMonthWeek` (472)

Listing: `./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);
  WriteIn('Today is : ',DateToStr(TS));
End.

```

9.4.24 DecodeDayOfWeekInMonth

Synopsis: Decode a DateTime 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` ([508](#)), `EncodeDateMonthWeek` ([475](#)), `#rtl.sysutils.DayOfWeek` ([1416](#)), `EncodeDayOfWeekInMonth` ([476](#)), `TryEncodeDayOfWeekInMonth` ([532](#))

Listing: `./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 ');
  WriteIn(formatDateTime('dddd',D),' of the month. ');
End.

```

9.4.25 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` ([465](#)) function.

See also: [DateTimeToDosDateTime \(465\)](#)

9.4.26 EncodeDateDay

Synopsis: Encodes a year and day of year to a `DateTime` 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 \(471\)](#).

Errors: If any of the arguments is not valid, then an `EConvertError` exception is raised.

See also: [EncodeDateMonthWeek \(475\)](#), [DecodeDateDay \(471\)](#), [EncodeDateTime \(475\)](#), [EncodeDateWeek \(476\)](#), [TryEncodeDateTime \(530\)](#), [TryEncodeDateMonthWeek \(529\)](#), [TryEncodeDateWeek \(531\)](#)

9.4.27 EncodeDateMonthWeek

Synopsis: Encodes a year, month, week of month and day of week to a `DateTime` value

Declaration: `function EncodeDateMonthWeek(const AYear: Word;const AMonth: Word;
const AWeekOfMonth: Word;
const ADayOfWeek: Word) : TDateTime`

Visibility: default

Description: `EncodeDateMonthWeek` encodes the values `AYear`, `AMonth`, `WeekOfMonth`, `ADayOfWeek`, to a date value and returns this value.

For an example, see [DecodeDateMonthWeek \(472\)](#).

Errors: If any of the arguments is not valid, then an `EConvertError` exception is raised.

See also: [DecodeDateMonthWeek \(472\)](#), [EncodeDateTime \(475\)](#), [EncodeDateWeek \(476\)](#), [EncodeDateDay \(475\)](#), [TryEncodeDateTime \(530\)](#), [TryEncodeDateWeek \(531\)](#), [TryEncodeDateMonthWeek \(529\)](#), [TryEncodeDateDay \(529\)](#), [NthDayOfWeek \(508\)](#)

9.4.28 EncodeDateTime

Synopsis: Encodes a `DateTime` 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 `AYear`, `AMonth`, `ADay`, `AHour`, `AMinute`, `ASecond` and `AMilliSecond` to a date/time value and returns this value.

For an example, see [DecodeDateTime \(473\)](#).

Errors: If any of the arguments is not valid, then an `EConvertError` exception is raised.

See also: [DecodeDateTime \(473\)](#), [EncodeDateMonthWeek \(475\)](#), [EncodeDateWeek \(476\)](#), [EncodeDateDay \(475\)](#), [TryEncodeDateTime \(530\)](#), [TryEncodeDateWeek \(531\)](#), [TryEncodeDateDay \(529\)](#), [TryEncodeDateMonthWeek \(529\)](#)

9.4.29 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 \(473\)](#).

Errors: If any of the arguments is not valid, then an `EConvertError` exception is raised.

See also: [EncodeDateMonthWeek \(475\)](#), [DecodeDateWeek \(473\)](#), [EncodeDateTime \(475\)](#), [EncodeDateDay \(475\)](#), [TryEncodeDateTime \(530\)](#), [TryEncodeDateWeek \(531\)](#), [TryEncodeDateMonthWeek \(529\)](#)

9.4.30 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 \(474\)](#).

Errors: If any of the values is not in range, then an `EConvertError` exception will be raised.

See also: [NthDayOfWeek \(508\)](#), [EncodeDateMonthWeek \(475\)](#), [#rtl.sysutils.DayOfWeek \(1416\)](#), [DecodeDayOfWeekInMonth \(474\)](#), [TryEncodeDayOfWeekInMonth \(532\)](#)

9.4.31 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` ([532](#))

9.4.32 EndOfDay

Synopsis: Calculates a `DateTime` 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` ([526](#)), `StartOfDay` ([523](#)), `StartOfTheWeek` ([527](#)), `StartOfAWeek` ([524](#)), `StartOfAMonth` ([524](#)), `StartOfTheMonth` ([526](#)), `EndOfTheWeek` ([480](#)), `EndOfAWeek` ([478](#)), `EndOfTheYear` ([481](#)), `EndOfAYear` ([479](#)), `EndOfTheMonth` ([480](#)), `EndOfAMonth` ([477](#)), `EndOfTheDay` ([479](#))

Listing: `./datutex/ex39.pp`

Program Example39;

{ This program demonstrates the EndOfDay function }

Uses SysUtils, DateUtils;

Const

Fmt = 'End of the day : "dd mmm 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.

9.4.33 EndOfAMonth

Synopsis: Calculate a datetime 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 \(526\)](#), [StartOfAMonth \(524\)](#), [EndOfTheMonth \(480\)](#), [EndOfTheYear \(481\)](#), [EndOfAYear \(479\)](#), [StartOfAWeek \(524\)](#), [StartOfTheWeek \(527\)](#)

Listing: ./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.

9.4.34 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 \(527\)](#), [EndOfTheWeek \(480\)](#), [EndOfAWeek \(478\)](#), [StartOfAMonth \(524\)](#), [EndOfTheYear \(481\)](#), [EndOfAYear \(479\)](#), [EndOfTheMonth \(480\)](#), [EndOfAMonth \(477\)](#)

Listing: ./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 );
WriteIn (FormatDateTime ( Fmt , EndOfAWeek ( Y , W ) ) );
WriteIn (FormatDateTime ( Fmt2 , EndOfAWeek ( Y , W , 6 ) ) );
End.

```

9.4.35 EndOfAYear

Synopsis: Calculate a DateTime value representing the last day of a year

Declaration: `function EndOfAYear(const AYear: Word) : TDateTime`

Visibility: default

Description: `StartOfAYear` returns a `TDateTime` value with the date of the last day of the year `AYear` (December 31).

See also: `StartOfTheYear` (527), `EndOfTheYear` (481), `EndOfAYear` (479), `EndOfTheMonth` (480), `EndOfA-Month` (477), `StartOfAWeek` (524), `StartOfTheWeek` (527)

Listing: `./datutex/ex27.pp`

Program Example27;

{ This program demonstrates the EndOfAYear function }

Uses SysUtils , DateUtils ;

Const

Fmt = 'Last day of this year : "dd mmm yyyy ';

Begin

WriteIn (FormatDateTime (Fmt , EndOfAYear (YearOf (Today))));

End.

9.4.36 EndOfTheDay

Synopsis: Calculate a datetime value that represents the end of a given day.

Declaration: `function EndOfTheDay(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: `EndOfTheDay` 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: `StartOfTheDay` (526), `StartOfADay` (523), `StartOfTheWeek` (527), `StartOfAWeek` (524), `StartOfA-Month` (524), `StartOfTheMonth` (526), `EndOfTheWeek` (480), `EndOfAWeek` (478), `EndOfTheYear` (481), `EndOfAYear` (479), `EndOfTheMonth` (480), `EndOfAMonth` (477), `EndOfADay` (477)

Listing: `./datutex/ex37.pp`

Program Example37;

{ This program demonstrates the EndOfTheDay function }

Uses SysUtils , DateUtils ;

Const

```
Fmt = ' "End of the day : "dd mmmm yyyy hh:nn:ss ';
```

Begin

```
WriteIn ( FormatDateTime ( Fmt , EndOfTheDay ( Today ) ) );
End.
```

9.4.37 EndOfTheMonth

Synopsis: Calculate a DateTime 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` (477) function.

See also: `StartOfAMonth` (524), `StartOfTheMonth` (526), `EndOfAMonth` (477), `EndOfTheYear` (481), `EndOfAYear` (479), `StartOfAWeek` (524), `StartOfTheWeek` (527)

Listing: `./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

```
WriteIn ( FormatDateTime ( Fmt , EndOfTheMonth ( Today ) ) );
End.
```

9.4.38 EndOfTheWeek

Synopsis: Calculate a DateTime 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` (478) function.

See also: `StartOfAWeek` (524), `StartOfTheWeek` (527), `EndOfAWeek` (478), `StartOfAMonth` (524), `EndOfTheYear` (481), `EndOfAYear` (479), `EndOfTheMonth` (480), `EndOfAMonth` (477)

Listing: `./datutex/ex33.pp`

Program Example33;

{ This program demonstrates the EndOfTheWeek function }

Uses SysUtils, DateUtils;

Const

Fmt = '"last day of this week : "dd mmm yyyy ';

Begin

WriteIn (FormatDateTime (Fmt, EndOfTheWeek (Today)));

End.

9.4.39 EndOfTheYear

Synopsis: Calculate a DateTime 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 (479) function.

See also: StartOfAYear (525), StartOfTheYear (527), EndOfTheMonth (480), EndOfAMonth (477), StartOfAWeek (524), StartOfTheWeek (527), EndOfAYear (479)

Listing: ./datutex/ex25.pp

Program Example25;

{ This program demonstrates the EndOfTheYear function }

Uses SysUtils, DateUtils;

Const

Fmt = '"Last day of this year : "dd mmm yyyy ';

Begin

WriteIn (FormatDateTime (Fmt, EndOfTheYear (Today)));

End.

9.4.40 HourOf

Synopsis: Extract the hour part from a DateTime 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 (547)

See also: YearOf (547), WeekOf (535), MonthOf (506), DayOf (466), MinuteOf (502), SecondOf (519), MilliSecondOf (497)

9.4.41 HourOfDay

Synopsis: Calculate the hour of a given `DateTime` value

Declaration: `function HourOfDay(const AValue: TDateTime) : Word`

Visibility: default

Description: `HourOfDay` 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: `HourOfYear` ([483](#)), `HourOfMonth` ([482](#)), `HourOfWeek` ([482](#)), `MinuteOfDay` ([502](#)), `SecondOfDay` ([519](#)), `MillisecondOfDay` ([498](#))

Listing: `./datutex/ex43.pp`

Program `Example43`;

{ This program demonstrates the HourOfDay function }

Uses `SysUtils, DateUtils`;

Var

`N : TDateTime`;

Begin

`N:=Now`;

`WriteLn('Hour of the Day : ', HourOfDay(N));`

`WriteLn('Minute of the Day : ', MinuteOfDay(N));`

`WriteLn('Second of the Day : ', SecondOfDay(N));`

`WriteLn('Millisecond of the Day : ',
MillisecondOfDay(N));`

End.

9.4.42 HourOfMonth

Synopsis: Calculate the number of hours passed since the start of the month.

Declaration: `function HourOfMonth(const AValue: TDateTime) : Word`

Visibility: default

Description: `HourOfMonth` 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 `WeekOfMonth` ([535](#)) function.

See also: `WeekOfMonth` ([535](#)), `DayOfMonth` ([466](#)), `MinuteOfMonth` ([503](#)), `SecondOfMonth` ([520](#)), `MillisecondOfMonth` ([499](#))

9.4.43 HourOfWeek

Synopsis: Calculate the number of hours elapsed since the start of the week.

Declaration: `function HourOfWeek(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` (467) function.

See also: `HourOfTheYear` (483), `HourOfTheMonth` (482), `HourOfTheDay` (482), `DayOfTheWeek` (467), `MinuteOfTheWeek` (503), `SecondOfTheWeek` (521), `MilliSecondOfTheWeek` (499)

9.4.44 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` (536) function.

See also: `WeekOfTheYear` (536), `DayOfTheYear` (467), `MinuteOfTheYear` (503), `SecondOfTheYear` (521), `MilliSecondOfTheYear` (500)

9.4.45 HoursBetween

Synopsis: Calculate the number of whole hours between two `DateTime` 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` (548), `MonthsBetween` (506), `WeeksBetween` (536), `DaysBetween` (468), `MinutesBetween` (504), `SecondsBetween` (521), `MillisecondsBetween` (500)

Listing: `./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.**9.4.46 HourSpan**

Synopsis: Calculate the approximate number of hours between two DateTime 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` (549), `MonthSpan` (507), `WeekSpan` (538), `DaySpan` (471), `MinuteSpan` (505), `SecondSpan` (522), `MilliSecondSpan` (501), `HoursBetween` (483)

Listing: `./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);

```

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.**

9.4.49 IncMilliSecond

Synopsis: Increase a DateTime 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 ([488](#)), #rtl.sysutils.IncMonth ([1468](#)), IncWeek ([487](#)), IncDay ([485](#)), IncHour ([485](#)), IncSecond ([487](#)), IncMilliSecond ([486](#))

Listing: ./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.**

9.4.50 IncMinute

Synopsis: Increase a DateTime 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 ([488](#)), #rtl.sysutils.IncMonth ([1468](#)), IncWeek ([487](#)), IncDay ([485](#)), IncHour ([485](#)), IncSecond ([487](#)), IncMilliSecond ([486](#))

Listing: ./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.

9.4.51 IncSecond

Synopsis: Increase a DateTime 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 ([488](#)), #rtl.sysutils.IncMonth ([1468](#)), IncWeek ([487](#)), IncDay ([485](#)), IncHour ([485](#)), IncSecond ([487](#)), IncMilliSecond ([486](#))

Listing: ./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.

9.4.52 IncWeek

Synopsis: Increase a DateTime 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 ([488](#)), #rtl.sysutils.IncMonth ([1468](#)), IncDay ([485](#)), IncHour ([485](#)), IncMinute ([486](#)), IncSecond ([487](#)), IncMilliSecond ([486](#))

Listing: ./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.

9.4.53 IncYear

Synopsis: Increase a DateTime 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 (1468), IncWeek (487), IncDay (485), IncHour (485), IncMinute (486), IncSecond (487), IncMilliSecond (486)

Listing: ./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.

9.4.54 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 (1521) 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 (489), InvalidDateTimeError (489), InvalidDateMonthWeekError (489), InvalidDayOfWeekInMonthError (490)

9.4.55 InvalidDateMonthWeekError

Synopsis: Raise an `EConvertError` exception when a `Year,Month,WeekOfMonth,DayOfWeek` is invalid.

Declaration:

```
procedure InvalidDateMonthWeekError(const AYear: Word;
                                     const AMonth: Word;
                                     const AWeekOfMonth: Word;
                                     const ADayOfWeek: Word)
```

Visibility: default

Description: `InvalidDateMonthWeekError` raises an `EConvertError` (1521) 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` (489), `InvalidDateTimeError` (489), `InvalidDateDayError` (488), `InvalidDay-Of-WeekInMonthError` (490)

9.4.56 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` (1521) 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` (489), `InvalidDateDayError` (488), `InvalidDateMonthWeekError` (489), `InvalidDayOf-WeekInMonthError` (490)

9.4.57 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` (1521) 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` (489), `InvalidDateDayError` (488), `InvalidDateMonthWeekError` (489), `InvalidDayOfWeekInMonthError` (490)

9.4.58 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` (1521) 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` (489), `InvalidDateTimeError` (489), `InvalidDateDayError` (488), `InvalidDateMonthWeekError` (489)

9.4.59 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` (547), `IsPM` (491), `IsToday` (492), `IsSameDay` (491)

Listing: `./datutex/ex3.pp`

Program Example3;

{ This program demonstrates the IsInLeapYear function }

Uses SysUtils, DateUtils;

Begin

WriteIn('Current year is leap year: ',IsInLeapYear(**Date**));

End.

9.4.60 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` (547), `IsInLeapYear` (490), `IsToday` (492), `IsSameDay` (491)

Listing: `./datutex/ex4.pp`

Program Example4;

{ This program demonstrates the IsPM function }

Uses SysUtils, DateUtils;

Begin

`WriteLn('Current time is PM : ', IsPM(Now));`

End.

9.4.61 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` (528), `Yesterday` (549), `Tomorrow` (528), `IsToday` (492)

Listing: `./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 mmm yyyy 'is today : ', D));`

`WriteLn(IsSameDay(D, Today));`

`end;`

End.

9.4.62 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` (491), `IsToday` (492), `SameDate` (516)

9.4.63 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` (528), `Yesterday` (549), `Tomorrow` (528), `IsSameDay` (491)

Listing: `./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.

9.4.64 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` (496), `IsValidDateTime` (494), `IsValidDateDay` (493), `IsValidDateWeek` (495), `IsValidDateMonthWeek` (493)

Listing: `./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.

```

9.4.65 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 ([492](#)), IsValidTime ([496](#)), IsValidDateTime ([494](#)), IsValidDateWeek ([495](#)), IsValidDateMonthWeek ([493](#))

Listing: ./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.

```

9.4.66 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` (492), `IsValidTime` (496), `IsValidDateTime` (494), `IsValidDateDay` (493), `IsValidDate-Week` (495)

Listing: `./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);`
 `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.

9.4.67 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` (492), `IsValidTime` (496), `IsValidDateDay` (493), `IsValidDateWeek` (495), `IsValidDate-MonthWeek` (493)

Listing: `./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.

9.4.68 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` (492), `IsValidTime` (496), `IsValidDateTime` (494), `IsValidDateDay` (493), `IsValidDate-MonthWeek` (493)

Listing: `./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;
End.

```

9.4.69 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.

9.4.70 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 ([465](#)), TryJulianDateToDateTime ([533](#)), DateTimeToModifiedJulianDate ([466](#)), TryModifiedJulianDateToDateTime ([533](#))

9.4.71 LocalTimeToUniversal

Synopsis: Convert local time to UTC time

Declaration: `function LocalTimeToUniversal(LT: TDateTime) : TDateTime`
`function LocalTimeToUniversal(LT: TDateTime; TZOffset: Integer)`
`: TDateTime`

Visibility: default

Description: `UniversalTimeToLocal` converts a local time indication to a universal time indication: it undoes the `TZOffset` timezone offset from the UT Universal time (UTC). If no `TZOffset` is specified, the local time offset as returned by `GetLocalTimeOffset` (459) is used.

See also: `GetLocalTimeOffset` (459), `UniversalTimeToLocal` (534)

9.4.72 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` (534), `DateTimeToMac` (465), `MacToDateTime` (497)

9.4.73 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` (534), `DateTimeToMac` (465), `MacTimeStampToUnix` (497)

9.4.74 MilliSecondOf

Synopsis: Extract the millisecond part from a DateTime 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` (547)

See also: `YearOf` (547), `WeekOf` (535), `MonthOf` (506), `DayOf` (466), `HourOf` (481), `MinuteOf` (502), `MilliSecondOf` (497)

9.4.75 MilliSecondOfDay

Synopsis: Calculate the number of milliseconds elapsed since the start of the day

Declaration: `function MilliSecondOfDay(const AValue: TDateTime) : LongWord`

Visibility: default

Description: `MilliSecondOfDay` 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 `HourOfDay` (482) function.

See also: `MilliSecondOfTheYear` (500), `MilliSecondOfTheMonth` (499), `MilliSecondOfTheWeek` (499), `MilliSecondOfTheHour` (498), `MilliSecondOfTheMinute` (498), `MilliSecondOfTheSecond` (499), `HourOfTheDay` (482), `MinuteOfDay` (502), `SecondOfDay` (519)

9.4.76 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` (502) function.

See also: `MilliSecondOfTheYear` (500), `MilliSecondOfTheMonth` (499), `MilliSecondOfTheWeek` (499), `MilliSecondOfDay` (498), `MilliSecondOfTheMinute` (498), `MilliSecondOfTheSecond` (499), `MinuteOfTheHour` (502), `SecondOfTheHour` (520)

9.4.77 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` (520) function.

See also: `MilliSecondOfTheYear` (500), `MilliSecondOfTheMonth` (499), `MilliSecondOfTheWeek` (499), `MilliSecondOfDay` (498), `MilliSecondOfTheHour` (498), `MilliSecondOfTheMinute` (498), `MilliSecondOfTheSecond` (499), `SecondOfTheMinute` (520)

9.4.78 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` (535) function.

See also: `WeekOfTheMonth` (535), `DayOfTheMonth` (466), `HourOfTheMonth` (482), `MinuteOfTheMonth` (503), `SecondOfTheMonth` (520), `MilliSecondOfTheMonth` (499)

9.4.79 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` (500), `MilliSecondOfTheMonth` (499), `MilliSecondOfTheWeek` (499), `MilliSecondOfTheDay` (498), `MilliSecondOfTheHour` (498), `MilliSecondOfTheMinute` (498), `SecondOfTheMinute` (520)

Listing: `./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.

9.4.80 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` (467) function.

See also: `MilliSecondOfTheYear` (500), `MilliSecondOfTheMonth` (499), `MilliSecondOfTheDay` (498), `MilliSecondOfTheHour` (498), `MilliSecondOfTheMinute` (498), `MilliSecondOfTheSecond` (499), `DayOfTheWeek` (467), `HourOfTheWeek` (482), `MinuteOfTheWeek` (503), `SecondOfTheWeek` (521)

9.4.81 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` (536) function.

See also: `WeekOfTheYear` (536), `DayOfTheYear` (467), `HourOfTheYear` (483), `MinuteOfTheYear` (503), `SecondOfTheYear` (521), `MilliSecondOfTheYear` (500)

9.4.82 MilliSecondsBetween

Synopsis: Calculate the number of whole milliseconds between two `DateTime` 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` (548), `MonthsBetween` (506), `WeeksBetween` (536), `DaysBetween` (468), `HoursBetween` (483), `MinutesBetween` (504), `SecondsBetween` (521)

Listing: `./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

Begin

End .

Begin

```
D2:=D1-(2.5*OneMilliSecond);
Test(D1,D2);
End.
```

9.4.84 MinuteOf

Synopsis: Extract the minute part from a `DateTime` 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` ([547](#))

See also: `YearOf` ([547](#)), `WeekOf` ([535](#)), `MonthOf` ([506](#)), `DayOf` ([466](#)), `HourOf` ([481](#)), `SecondOf` ([519](#)), `MilliSecondOf` ([497](#))

9.4.85 MinuteOfDay

Synopsis: Calculate the number of minutes elapsed since the start of the day

Declaration: `function MinuteOfDay(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` ([482](#)) function.

See also: `MinuteOfYear` ([503](#)), `MinuteOfMonth` ([503](#)), `MinuteOfWeek` ([503](#)), `MinuteOfTheHour` ([502](#)), `HourOfDay` ([482](#)), `SecondOfDay` ([519](#)), `MilliSecondOfDay` ([498](#))

9.4.86 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` ([503](#)), `MinuteOfMonth` ([503](#)), `MinuteOfWeek` ([503](#)), `MinuteOfDay` ([502](#)), `SecondOfTheHour` ([520](#)), `MilliSecondOfTheHour` ([498](#))

Listing: `./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.

9.4.87 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 (535) function.

See also: WeekOfTheMonth (535), DayOfTheMonth (466), HourOfTheMonth (482), MinuteOfTheMonth (503), SecondOfTheMonth (520), MilliSecondOfTheMonth (499)

9.4.88 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 (467) function.

See also: MinuteOfTheYear (503), MinuteOfTheMonth (503), MinuteOfTheDay (502), MinuteOfTheHour (502), DayOfTheWeek (467), HourOfTheWeek (482), SecondOfTheWeek (521), MilliSecondOfTheWeek (499)

9.4.89 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` (536) function.

See also: `WeekOfTheYear` (536), `DayOfTheYear` (467), `HourOfTheYear` (483), `MinuteOfTheYear` (503), `SecondOfTheYear` (521), `MilliSecondOfTheYear` (500)

9.4.90 MinutesBetween

Synopsis: Calculate the number of whole minutes between two `DateTime` 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` (548), `MonthsBetween` (506), `WeeksBetween` (536), `DaysBetween` (468), `HoursBetween` (483), `SecondsBetween` (521), `MillisecondsBetween` (500)

Listing: `./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.

9.4.91 MinuteSpan

Synopsis: Calculate the approximate number of minutes between two `DateTime` 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` (549), `MonthSpan` (507), `WeekSpan` (538), `DaySpan` (471), `HourSpan` (484), `SecondSpan` (522), `MilliSecondSpan` (501), `MinutesBetween` (504)

Listing: `./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.

9.4.92 ModifiedJulianDateToDateTime

Synopsis: Convert a modified Julian date representation to a `TDateTime` value.

Declaration: `function ModifiedJulianDateToDateTime(const AValue: Double) : TDateTime`

Visibility: default

Description: Not yet implemented.

Errors: Currently, trying to use this function will raise an exception.

See also: [DateTimeToJulianDate \(465\)](#), [JulianDateToDateTime \(496\)](#), [TryJulianDateToDateTime \(533\)](#), [DateTimeToModifiedJulianDate \(466\)](#), [TryModifiedJulianDateToDateTime \(533\)](#)

9.4.93 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 \(547\)](#)

See also: [YearOf \(547\)](#), [DayOf \(466\)](#), [WeekOf \(535\)](#), [HourOf \(481\)](#), [MinuteOf \(502\)](#), [SecondOf \(519\)](#), [MilliSecondOf \(497\)](#)

9.4.94 MonthOfTheYear

Synopsis: Extract the month of a `DateTime` 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 \(506\)](#), and is provided for completeness only, corresponding to the other `PartOfTheYear` functions.

For an example, see the [WeekOfTheYear \(536\)](#) function.

See also: [MonthOf \(506\)](#), [WeekOfTheYear \(536\)](#), [DayOfTheYear \(467\)](#), [HourOfTheYear \(483\)](#), [MinuteOfTheYear \(503\)](#), [SecondOfTheYear \(521\)](#), [MilliSecondOfTheYear \(500\)](#)

9.4.95 MonthsBetween

Synopsis: Calculate the number of whole months between two `DateTime` values

Declaration: `function MonthsBetween(const ANow: TDateTime; const AThen: TDateTime) : 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 \(548\)](#), [WeeksBetween \(536\)](#), [DaysBetween \(468\)](#), [HoursBetween \(483\)](#), [MinutesBetween \(504\)](#), [SecondsBetween \(521\)](#), [MilliSecondsBetween \(500\)](#)

Listing: `./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.

```

9.4.96 MonthSpan

Synopsis: Calculate the approximate number of months between two DateTime 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 \(549\)](#), [WeekSpan \(538\)](#), [DaySpan \(471\)](#), [HourSpan \(484\)](#), [MinuteSpan \(505\)](#), [SecondSpan \(522\)](#), [MilliSecondSpan \(501\)](#), [MonthsBetween \(506\)](#)

Listing: ./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.

```

9.4.97 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` ([475](#)), `#rtl.sysutils.DayOfWeek` ([1416](#)), `DecodeDayOfWeekInMonth` ([474](#)), `EncodeDayOfWeekInMonth` ([476](#)), `TryEncodeDayOfWeekInMonth` ([532](#))

Listing: `./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.

```

9.4.98 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` (548), `MonthsBetween` (506), `WeeksBetween` (536), `DaysBetween` (468)

9.4.99 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` (549)

Listing: `./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.

9.4.100 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 \(515\)](#), [RecodeMonth \(513\)](#), [RecodeDay \(511\)](#), [RecodeHour \(511\)](#), [RecodeMinute \(513\)](#), [RecodeSecond \(514\)](#), [RecodeDate \(509\)](#), [RecodeTime \(515\)](#), [RecodeDateTime \(510\)](#)

Listing: ./datutex/ex94.pp

Program Example94;

{ This program demonstrates the RecodeDate function }

Uses SysUtils, DateUtils;

Const

 Fmt = 'dddd dd mmm yyyy hh:nn:ss';

Var

 S : AnsiString;

Begin

 S := **FormatDateTime**(Fmt, **RecodeDate**(**Now**, 2001, 1, 1));

WriteIn('This moment on the first of the millenium : ', S);

End.

9.4.101 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 pre-defined constant `RecodeLeaveFieldAsIs` ([461](#)), then the corresponding part of the date/time stamp is left untouched.

Errors: If one of the values `AYear`, `AMonth`, `ADay`, `AHour`, `AMinute`, `ASecond`, `AMilliSecond` is not within a valid range (`RecodeLeaveFieldAsIs` excepted) then an `EConvertError` exception is raised.

See also: [RecodeYear \(515\)](#), [RecodeMonth \(513\)](#), [RecodeDay \(511\)](#), [RecodeHour \(511\)](#), [RecodeMinute \(513\)](#), [RecodeSecond \(514\)](#), [RecodeMilliSecond \(512\)](#), [RecodeDate \(509\)](#), [RecodeTime \(515\)](#), [TryRecodeDateTime \(533\)](#)

Listing: ./datutex/ex96.pp

Program Example96;

{ This program demonstrates the RecodeDateTime function }

Uses SysUtils, DateUtils;

```

Const
  Fmt = 'dddd dd mmm 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.

```

9.4.102 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 (515), RecodeMonth (513), RecodeHour (511), RecodeMinute (513), RecodeSecond (514), RecodeMilliSecond (512), RecodeDate (509), RecodeTime (515), RecodeDateTime (510)

Listing: ./datutex/ex89.pp

Program Example89;

{ This program demonstrates the RecodeDay function }

Uses SysUtils, DateUtils;

```

Const
  Fmt = 'dddd dd mmm yyyy hh:nn:ss';

```

```

Var
  S : AnsiString;

```

```

Begin
  S:=FormatDateTime(Fmt,RecodeDay(Now,1));
  WriteLn('This moment on the first of the month : ',S);
End.

```

9.4.103 RecodeHour

Synopsis: Replace hours part of a TDateTime value with another hour.

Declaration: `function RecodeHour(const AValue: TDateTime;const AHour: Word) : TDateTime`

Visibility: default

Description: `RecodeHour` replaces the Hour part of the timestamp `AValue` with `AHour`. All other parts of the date/time stamp are left untouched.

Errors: If the `AHour` value is not within a valid range (0..23) then an `EConvertError` exception is raised.

See also: `RecodeYear` (515), `RecodeMonth` (513), `RecodeDay` (511), `RecodeMinute` (513), `RecodeSecond` (514), `RecodeMilliSecond` (512), `RecodeDate` (509), `RecodeTime` (515), `RecodeDateTime` (510)

Listing: `./datutex/ex90.pp`

Program `Example90`;

{ This program demonstrates the RecodeHour function }

Uses `SysUtils`, `DateUtils`;

Const

`Fmt = 'dddd dd mmm yyyy hh:nn:ss';`

Var

`S : AnsiString;`

Begin

`S:=FormatDateTime(Fmt,RecodeHour(Now,0));`

`WriteLn('Today, in the first hour : ',S);`

End.

9.4.104 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` (515), `RecodeMonth` (513), `RecodeDay` (511), `RecodeHour` (511), `RecodeMinute` (513), `RecodeSecond` (514), `RecodeDate` (509), `RecodeTime` (515), `RecodeDateTime` (510)

Listing: `./datutex/ex93.pp`

Program `Example93`;

{ This program demonstrates the RecodeMilliSecond function }

Uses `SysUtils`, `DateUtils`;

Const

`Fmt = 'dddd dd mmm yyyy hh:nn:ss.zzz';`

```

Var
  S : AnsiString;

Begin
  S:=FormatDateTime(Fmt,RecodeMilliSecond(Now,0));
  WriteLn('This moment, milliseconds stripped : ',S);
End.

```

9.4.105 RecodeMinute

Synopsis: Replace minutse 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 (515), RecodeMonth (513), RecodeDay (511), RecodeHour (511), RecodeSecond (514), RecodeMilliSecond (512), RecodeDate (509), RecodeTime (515), RecodeDateTime (510)

Listing: ./datutex/ex91.pp

Program Example91;

{ This program demonstrates the RecodeMinute function }

Uses SysUtils, DateUtils;

Const

Fmt = 'dddd dd mmm 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.

9.4.106 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 \(515\)](#), [RecodeDay \(511\)](#), [RecodeHour \(511\)](#), [RecodeMinute \(513\)](#), [RecodeSecond \(514\)](#), [RecodeMilliSecond \(512\)](#), [RecodeDate \(509\)](#), [RecodeTime \(515\)](#), [RecodeDateTime \(510\)](#)

Listing: ./datutex/ex88.pp

Program Example88;

{ This program demonstrates the RecodeMonth function }

Uses SysUtils, DateUtils;

Const

 Fmt = 'dddd dd mmm yyyy hh:nn:ss';

Var

 S : AnsiString;

Begin

 S := **FormatDateTime**(Fmt, RecodeMonth(**Now**, 5));

WriteIn('This moment in May : ', S);

End.

9.4.107 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 \(515\)](#), [RecodeMonth \(513\)](#), [RecodeDay \(511\)](#), [RecodeHour \(511\)](#), [RecodeMinute \(513\)](#), [RecodeMilliSecond \(512\)](#), [RecodeDate \(509\)](#), [RecodeTime \(515\)](#), [RecodeDateTime \(510\)](#)

Listing: ./datutex/ex92.pp

Program Example92;

{ This program demonstrates the RecodeSecond function }

Uses SysUtils, DateUtils;

Const

 Fmt = 'dddd dd mmm yyyy hh:nn:ss';

Var

 S : AnsiString;

```

Begin
  S:=FormatDateTime(Fmt,RecodeSecond(Now,0));
  WriteLn('This moment, seconds stripped : ',S);
End.

```

9.4.108 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, ASecondAMilliSecond is not within a valid range then an EConvertError exception is raised.

See also: RecodeYear (515), RecodeMonth (513), RecodeDay (511), RecodeHour (511), RecodeMinute (513), RecodeSecond (514), RecodeMilliSecond (512), RecodeDate (509), RecodeDateTime (510)

Listing: ./datutex/ex95.pp

Program Example95;

{ This program demonstrates the RecodeTime function }

Uses SysUtils, DateUtils;

Const
 Fmt = 'dddd dd mmm yyyy hh:nn:ss';

Var
 S : AnsiString;

Begin
 S:=**FormatDateTime**(Fmt,RecodeTime(**Now**,8,0,0,0));
 WriteLn('Today, 8 AM : ',S);
End.

9.4.109 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 \(513\)](#), [RecodeDay \(511\)](#), [RecodeHour \(511\)](#), [RecodeMinute \(513\)](#), [RecodeSecond \(514\)](#), [RecodeMilliSecond \(512\)](#), [RecodeDate \(509\)](#), [RecodeTime \(515\)](#), [RecodeDateTime \(510\)](#)

Listing: ./datutex/ex87.pp

Program Example87;

{ This program demonstrates the RecodeYear function }

Uses SysUtils, DateUtils;

Const

 Fmt = 'dddd dd mmm yyyy hh:nn:ss';

Var

 S : AnsiString;

Begin

 S:=**FormatDateTime**(Fmt,RecodeYear(**Now**,1999));

WriteIn('This moment in 1999 : ',S);

End.

9.4.110 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` ([461](#)) returns zero.

See also: [CompareDateTime \(462\)](#), [CompareDate \(461\)](#), [CompareTime \(463\)](#), [SameDateTime \(517\)](#), [SameTime \(518\)](#)

Listing: ./datutex/ex102.pp

Program Example102;

{ This program demonstrates the SameDate function }

Uses SysUtils, DateUtils;

Const

 Fmt = 'dddd dd mmm yyyy hh:nn:ss.zzz';

Procedure Test(D1,D2 : TDateTime);

begin

Write(**FormatDateTime**(Fmt,D1),' is the same date as ');

WriteIn(**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.

```

9.4.111 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 (462) returns zero.

See also: CompareDateTime (462), CompareDate (461), CompareTime (463), SameDate (516), SameTime (518)

Listing: ./datutex/ex101.pp

Program Example101;

{ This program demonstrates the SameDateTime function }

Uses SysUtils, DateUtils;

Const

 Fmt = 'dddd dd mmm yyyy hh:nn:ss.zzz';

Procedure Test(D1,D2 : TDateTime);

begin

Write(FormatDateTime(Fmt,D1), ' is the same datetime as ');

WriteIn(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.

```

9.4.112 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` (463) returns zero.

See also: `CompareDateTime` (462), `CompareDate` (461), `CompareTime` (463), `SameDateTime` (517), `SameDate` (516)

Listing: `./datutex/ex103.pp`

Program Example102;

{ This program demonstrates the SameTime function }

Uses SysUtils, DateUtils;

Const

 Fmt = 'dddd dd mmm 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.

9.4.113 ScanDateTime

Synopsis: Scans a string for a `DateTime` 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` (1457) 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.

Errors: In case of an error, a `EConvertError` (1521) exception is raised.

See also: `FormatDateTime` (1457)

9.4.114 SecondOf

Synopsis: Extract the second part from a `DateTime` 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` (547)

See also: `YearOf` (547), `WeekOf` (535), `MonthOf` (506), `DayOf` (466), `HourOf` (481), `MinuteOf` (502), `MilliSecondOf` (497)

9.4.115 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` (482) function.

See also: `SecondOfTheYear` (521), `SecondOfTheMonth` (520), `SecondOfTheWeek` (521), `SecondOfTheHour` (520), `SecondOfTheMinute` (520), `HourOfDay` (482), `MinuteOfDay` (502), `MilliSecondOfTheDay` (498)

9.4.116 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` (502) function.

See also: `SecondOfTheYear` (521), `SecondOfTheMonth` (520), `SecondOfTheWeek` (521), `SecondOfTheDay` (519), `SecondOfTheMinute` (520), `MinuteOfTheHour` (502), `MilliSecondOfTheHour` (498)

9.4.117 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` (521), `SecondOfTheMonth` (520), `SecondOfTheWeek` (521), `SecondOfTheDay` (519), `SecondOfTheHour` (520), `MilliSecondOfTheMinute` (498)

Listing: `./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.

9.4.118 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` (535) function.

See also: `WeekOfTheMonth` (535), `DayOfTheMonth` (466), `HourOfTheMonth` (482), `MinuteOfTheMonth` (503), `MilliSecondOfTheMonth` (499)

9.4.119 `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` (467) function.

See also: `SecondOfTheYear` (521), `SecondOfTheMonth` (520), `SecondOfTheDay` (519), `SecondOfTheHour` (520), `SecondOfTheMinute` (520), `DayOfTheWeek` (467), `HourOfTheWeek` (482), `MinuteOfTheWeek` (503), `MilliSecondOfTheWeek` (499)

9.4.120 `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` (536) function.

See also: `WeekOfTheYear` (536), `DayOfTheYear` (467), `HourOfTheYear` (483), `MinuteOfTheYear` (503), `SecondOfTheYear` (521), `MilliSecondOfTheYear` (500)

9.4.121 `SecondsBetween`

Synopsis: Calculate the number of whole seconds between two `DateTime` 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` (548), `MonthsBetween` (506), `WeeksBetween` (536), `DaysBetween` (468), `HoursBetween` (483), `MinutesBetween` (504), `MillisecondsBetween` (500)


```

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.

```

9.4.123 StartOfDay

Synopsis: Return the start of a day as a DateTime 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: StartOfTheDay ([526](#)), StartOfTheWeek ([527](#)), StartOfAWeek ([524](#)), StartOfAMonth ([524](#)), StartOfTheMonth ([526](#)), EndOfTheWeek ([480](#)), EndOfAWeek ([478](#)), EndOfTheYear ([481](#)), EndOfAYear ([479](#)), EndOfTheMonth ([480](#)), EndOfAMonth ([477](#)), EndOfTheDay ([479](#)), EndOfDay ([477](#))

Listing: ./datutex/ex38.pp

Program Example38;

{ This program demonstrates the StartOfDay function }

Uses SysUtils, DateUtils;

Const

Fmt = ' "Start of the day : " dd mmm yyyy hh:nn:ss ';


```

Var
  Y,M,D : Word;

Begin
  Y:=YearOf(Today);
  M:=MonthOf(Today);
  D:=DayOf(Today);
  WriteIn(FormatDateTime(Fmt, StartOfADay(Y,M,D)));
  DecodeDateDay(Today,Y,D);
  WriteIn(FormatDateTime(Fmt, StartOfADay(Y,D)));
End.

```

9.4.124 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` (526), `EndOfTheMonth` (480), `EndOfAMonth` (477), `EndOfTheYear` (481), `EndOfAYear` (479), `StartOfAWeek` (524), `StartOfTheWeek` (527)

Listing: ./datutex/ex30.pp

Program Example30;

{ This program demonstrates the StartOfAMonth function }

Uses SysUtils, DateUtils;

Const

Fmt = 'First day of this month : "dd mmm yyyy';

Var

Y,M : Word;

Begin

Y:=YearOf(Today);

M:=MonthOf(Today);

WriteIn(**FormatDateTime**(Fmt, StartOfAMonth(Y,M)));

End.

9.4.125 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` (527), `EndOfTheWeek` (480), `EndOfAWeek` (478), `StartOfAMonth` (524), `EndOfTheYear` (481), `EndOfAYear` (479), `EndOfTheMonth` (480), `EndOfAMonth` (477)

Listing: `./datutex/ex34.pp`

Program `Example34`;

{ This program demonstrates the StartOfAWeek function }

Uses `SysUtils`, `DateUtils`;

Const

`Fmt = ' "First day of this week : "dd mmm yyyy hh:nn:ss';`
`Fmt2 = ' "Second day of this week : "dd mmm 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.

9.4.126 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` (527), `EndOfTheYear` (481), `EndOfAYear` (479), `EndOfTheMonth` (480), `EndOfAMonth` (477), `StartOfAWeek` (524), `StartOfTheWeek` (527)

Listing: `./datutex/ex26.pp`

Program `Example26`;

{ This program demonstrates the StartOfAYear function }

Uses `SysUtils`, `DateUtils`;

Const

`Fmt = ' "First day of this year : "dd mmm yyyy';`

Begin

`WriteLn(FormatDateTime(Fmt, StartOfAYear(YearOf(Today))));`

End.

9.4.127 StartOfDay

Synopsis: Calculate the start of the day as a DateTime value, given a moment in the day.

Declaration: `function StartOfDay(const AValue: TDateTime) : TDateTime`

Visibility: default

Description: `StartOfDay` 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: `StartOfDay` (523), `StartOfTheWeek` (527), `StartOfAWeek` (524), `StartOfAMonth` (524), `StartOfTheMonth` (526), `EndOfTheWeek` (480), `EndOfAWeek` (478), `EndOfTheYear` (481), `EndOfAYear` (479), `EndOfTheMonth` (480), `EndOfAMonth` (477), `EndOftheDay` (479), `EndOfADay` (477)

Listing: `./datutex/ex36.pp`

Program `Example36;`

{ This program demonstrates the StartOfDay function }

Uses `SysUtils, DateUtils;`

Const

`Fmt = ' "Start of the day : "dd mmm yyyy hh:nn:ss ';`

Begin

`WriteIn (FormatDateTime (Fmt, StartOfDay (Today)));`

End.

9.4.128 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` (524) function.

See also: `StartOfAMonth` (524), `EndOfTheYear` (481), `EndOfAYear` (479), `EndOfTheMonth` (480), `EndOfAMonth` (477), `StartOfAWeek` (524), `StartOfTheWeek` (527)

Listing: `./datutex/ex28.pp`

Program `Example28;`

{ This program demonstrates the StartOfTheMonth function }

Uses `SysUtils, DateUtils;`

Const

`Fmt = ' "First day of this month : "dd mmm yyyy ';`

Begin

`WriteIn (FormatDateTime (Fmt, StartOfTheMonth (Today)));`

End.

9.4.129 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` (524) function.

See also: `StartOfAWeek` (524), `EndOfTheWeek` (480), `EndOfAWeek` (478), `StartOfAMonth` (524), `EndOfTheYear` (481), `EndOfAYear` (479), `EndOfTheMonth` (480), `EndOfAMonth` (477)

Listing: `./datutex/ex32.pp`

Program `Example32;`

{ This program demonstrates the StartOfTheWeek function }

Uses `SysUtils, DateUtils;`

Const

`Fmt = ' "First day of this week : "dd mmm yyyy ';`

Begin

`WriteIn (FormatDateTime (Fmt, StartOfTheWeek (Today)));`

End.

9.4.130 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` (525) function.

See also: `StartOfAYear` (525), `EndOfTheYear` (481), `EndOfAYear` (479)

Listing: `./datutex/ex24.pp`

Program `Example24;`

{ This program demonstrates the StartOfTheYear function }

Uses `SysUtils, DateUtils;`

Const

`Fmt = ' "First day of this year : "dd mmm yyyy ';`

Begin

`WriteIn (FormatDateTime (Fmt, StartOfTheYear (Today)));`

End.

9.4.131 TimeOf

Synopsis: Extract the time part from a DateTime 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` (464), `YearOf` (547), `MonthOf` (506), `DayOf` (466), `HourOf` (481), `MinuteOf` (502), `SecondOf` (519), `MilliSecondOf` (497)

Listing: `./datutex/ex2.pp`

Program `Example2;`

{ This program demonstrates the TimeOf function }

Uses `SysUtils, DateUtils;`

Begin

`WriteLn('Time is : ', TimeToStr(TimeOf(Now)));`
End.

9.4.132 Today

Synopsis: Return the current date

Declaration: `function Today : TDateTime`

Visibility: default

Description: `Today` is an alias for the `Date` (1412) function in the `sysutils` (1360) unit.

For an example, see `Yesterday` (549)

See also: `Date` (1412), `Yesterday` (549), `Tomorrow` (528)

9.4.133 Tomorrow

Synopsis: Return the next day

Declaration: `function Tomorrow : TDateTime`

Visibility: default

Description: `Tomorrow` returns tomorrow's date. `Tomorrow` is determined from the system clock, i.e. it is `Today` (528) +1.

See also: `Today` (528), `Yesterday` (549)

Listing: `./datutex/ex19.pp`

Program Example19;

{ This program demonstrates the Tomorrow function }

Uses SysUtils, DateUtils;

Begin

WriteLn (**FormatDateTime** (' "Today is " dd mmm yyyy ', Today));

WriteLn (**FormatDateTime** (' "Tomorrow will be " dd mmm yyyy ', Tomorrow));

End.

9.4.134 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 ([475](#)), EncodeDateTime ([475](#)), EncodeDateMonthWeek ([475](#)), EncodeDateWeek ([476](#)), TryEncodeDateTime ([530](#)), TryEncodeDateMonthWeek ([529](#)), TryEncodeDateWeek ([531](#))

Listing: ./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.

9.4.135 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: TryEncodeDateTime encodes the values AYearAMonth, WeekOfMonth, ADayOfWeek, to a date value and returns this value in AValue.

If the encoding was succesful, True is returned, False if any of the arguments is not valid.

See also: DecodeDateMonthWeek (472), EncodeDateTime (475), EncodeDateWeek (476), EncodeDateDay (475), EncodeDateMonthWeek (475), TryEncodeDateTime (530), TryEncodeDateWeek (531), TryEncodeDateDay (529), NthDayOfWeek (508)

Listing: ./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.

9.4.136 TryEncodeDateTime

Synopsis: Encode a Year, Month, Day, Hour, minute, seconds, milliseconds tuplet 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 succesfully, True is returned, False is returned if one of the arguments is not valid.

See also: EncodeDateTime (475), EncodeDateMonthWeek (475), EncodeDateWeek (476), EncodeDateDay (475), TryEncodeDateDay (529), TryEncodeDateWeek (531), TryEncodeDateMonthWeek (529)

Listing: ./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.

9.4.137 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 succesful, True is returned. False is returned if any of the arguments is not valid.

See also: EncodeDateMonthWeek ([475](#)), EncodeDateWeek ([476](#)), EncodeDateTime ([475](#)), EncodeDateDay ([475](#)), TryEncodeDateTime ([530](#)), TryEncodeDateMonthWeek ([529](#)), TryEncodeDateDay ([529](#))

Listing: ./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.

9.4.138 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 succesful, `False` if any of the values is not in range.

See also: `NthDayOfWeek` (508), `EncodeDateMonthWeek` (475), `#rtl.sysutils.DayOfWeek` (1416), `DecodeDayOfWeekInMonth` (474), `EncodeDayOfWeekInMonth` (476)

Listing: `./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.

9.4.139 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` (476)

9.4.140 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` (465), `JulianDateToDateTime` (496), `DateTimeToModifiedJulianDate` (466), `TryModifiedJulianDateToDateTime` (533)

9.4.141 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: Not yet implemented.

Errors: Currently, trying to use this function will raise an exception.

See also: `DateTimeToJulianDate` (465), `JulianDateToDateTime` (496), `TryJulianDateToDateTime` (533), `DateTimeToModifiedJulianDate` (466), `ModifiedJulianDateToDateTime` (505)

9.4.142 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 pre-defined constant `RecodeLeaveFieldAsIs` (461), 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` (515), `RecodeMonth` (513), `RecodeDay` (511), `RecodeHour` (511), `RecodeMinute` (513), `RecodeSecond` (514), `RecodeMilliSecond` (512), `RecodeDate` (509), `RecodeTime` (515), `RecodeDateTime` (510)

Listing: `./datutex/ex97.pp`

```

Program Example97;

{ This program demonstrates the TryRecodeDateTime function }

Uses SysUtils, DateUtils;

Const
    Fmt = 'dddd dd mmm yyyy hh:nn:ss';

Var
    S : AnsiString;
    D : TDateTime;

Begin
    If TryRecodeDateTime(Now, 2000, 2, RecodeLeaveFieldAsIs, 0, 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.

```

9.4.143 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` timezone offset to the UT Universal time (UTC). If no `TZOffset` is specified, the local time offset as returned by `GetLocalTimeOffset` (459) is used.

See also: `GetLocalTimeOffset` (459), `LocalTimeToUniversal` (497)

9.4.144 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` (465), `MacToDateTime` (497), `MacTimeStampToUnix` (497)

9.4.145 UnixToDateTime

Synopsis: Convert Unix epoch time to a `TDateTime` value

Declaration: `function UnixToDateTime(const AValue: Int64) : TDateTime`

Visibility: default

Description: `UnixToDateTime` converts epoch time (seconds elapsed since 1/1/1970) to a `TDateTime` value.

See also: `DateTimeToUnix` (466)

9.4.146 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` (547)

See also: `YearOf` (547), `DayOf` (466), `MonthOf` (506), `HourOf` (481), `MinuteOf` (502), `SecondOf` (519), `MilliSecondOf` (497)

9.4.147 WeekOfTheMonth

Synopsis: Extract the week of the month (and optionally month and year) from a `DateTime` 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` (536), `DayOfTheMonth` (466), `HourOfTheMonth` (482), `MinuteOfTheMonth` (503), `SecondOfTheMonth` (520), `MilliSecondOfTheMonth` (499)

Listing: `./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.

```

9.4.148 WeekOfTheYear

Synopsis: Extract the week of the year (and optionally year) of a `DateTime` 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` ([535](#)).

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` ([535](#)), `MonthOfTheYear` ([506](#)), `DayOfTheYear` ([467](#)), `HourOfTheYear` ([483](#)), `MinuteOfTheYear` ([503](#)), `SecondOfTheYear` ([521](#)), `MilliSecondOfTheYear` ([500](#))

Listing: `./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.

9.4.149 WeeksBetween

Synopsis: Calculate the number of whole weeks between two `DateTime` 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` (548), `MonthsBetween` (506), `DaysBetween` (468), `HoursBetween` (483), `MinutesBetween` (504), `SecondsBetween` (521), `MillisecondsBetween` (500)

Listing: `./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.

9.4.150 WeeksInAYear

Synopsis: Return the number of weeks in a given year

Declaration: `function WeeksInAYear(const AYear: Word) : Word`

Visibility: default

Description: `WeeksInAYear` 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 \(538\)](#), [DaysInYear \(470\)](#), [DaysInAYear \(469\)](#), [DaysInMonth \(470\)](#), [DaysInAMonth \(468\)](#)

Listing: ./datutex/ex13.pp

Program Example13;

{ This program demonstrates the WeeksInAYear function }

Uses SysUtils , DateUtils ;

Var

Y : Word;

Begin

For Y:=1992 to 2010 do

WriteLn(Y, ' has ', WeeksInAYear(Y), ' weeks. ');

End.

9.4.151 WeeksInYear

Synopsis: return 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 \(537\)](#), [DaysInYear \(470\)](#), [DaysInAYear \(469\)](#), [DaysInMonth \(470\)](#), [DaysInAMonth \(468\)](#)

Listing: ./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.

9.4.152 WeekSpan

Synopsis: Calculate the approximate number of weeks between two DateTime 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` (549), `MonthSpan` (507), `DaySpan` (471), `HourSpan` (484), `MinuteSpan` (505), `SecondSpan` (522), `MilliSecondSpan` (501), `WeeksBetween` (536)

Listing: `./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.
```

9.4.153 WithinPastDays

Synopsis: Check whether two datetimes 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` (468) 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 \(546\)](#), [WithinPastMonths \(543\)](#), [WithinPastWeeks \(545\)](#), [WithinPastHours \(540\)](#), [WithinPastMinutes \(542\)](#), [WithinPastSeconds \(544\)](#), [WithinPastMilliseconds \(541\)](#)

Listing: ./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.

9.4.154 WithinPastHours

Synopsis: Check whether two datetimes 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 \(483\)](#) 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 \(546\)](#), [WithinPastMonths \(543\)](#), [WithinPastWeeks \(545\)](#), [WithinPastDays \(539\)](#), [WithinPastMinutes \(542\)](#), [WithinPastSeconds \(544\)](#), [WithinPastMilliseconds \(541\)](#)

Listing: ./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.

9.4.155 WithinPastMilliseconds

Synopsis: Check whether two datetimes 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` (500) 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` (546), `WithinPastMonths` (543), `WithinPastWeeks` (545), `WithinPastDays` (539), `WithinPastHours` (540), `WithinPastMinutes` (542), `WithinPastSeconds` (544)

Listing: ./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.

9.4.156 WithinPastMinutes

Synopsis: Check whether two datetimes 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 (504) 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 (546), WithinPastMonths (543), WithinPastWeeks (545), WithinPastDays (539), WithinPastHours (540), WithinPastSeconds (544), WithinPastMilliseconds (541)

Listing: ./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.

9.4.157 WithinPastMonths

Synopsis: Check whether two datetimes 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 (506) 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 (546), WithinPastWeeks (545), WithinPastDays (539), WithinPastHours (540), WithinPastMinutes (542), WithinPastSeconds (544), WithinPastMilliseconds (541)

Listing: ./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.
```

9.4.158 WithinPastSeconds

Synopsis: Check whether two datetimes 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 (521) 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 (546), WithinPastMonths (543), WithinPastWeeks (545), WithinPastDays (539), WithinPastHours (540), WithinPastMinutes (542), WithinPastMilliseconds (541)

Listing: ./datutex/ex53.pp

Program Example53;

```
{ This program demonstrates the WithinPastSeconds function }
```

Uses SysUtils, DateUtils;

Procedure Test(ANow, AThen : TDateTime; ASeconds : Integer);

```

begin
  Write (DateTimeToStr(AThen), ' and ', DateTimeToStr(ANow));
  Write(' are within ', ASeconds, ' seconds: ');
  WriteLn (WithinPastSeconds(ANow, AThen, 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.

```

9.4.159 WithinPastWeeks

Synopsis: Check whether two datetimes 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` (536) 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` (546), `WithinPastMonths` (543), `WithinPastDays` (539), `WithinPastHours` (540), `WithinPastMinutes` (542), `WithinPastSeconds` (544), `WithinPastMilliseconds` (541)

Listing: `./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.

```

9.4.160 WithinPastYears

Synopsis: Check whether two datetimes 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` (548) 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` (543), `WithinPastWeeks` (545), `WithinPastDays` (539), `WithinPastHours` (540), `WithinPastMinutes` (542), `WithinPastSeconds` (544), `WithinPastMilliseconds` (541)

Listing: `./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.

```

9.4.161 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` ([506](#)), `DayOf` ([466](#)), `WeekOf` ([535](#)), `HourOf` ([481](#)), `MinuteOf` ([502](#)), `SecondOf` ([519](#)), `MilliSecondOf` ([497](#))

Listing: `./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.
```

9.4.162 YearsBetween

Synopsis: Calculate the number of whole years between two DateTime values

Declaration: `function YearsBetween(const ANow: TDateTime; const AThen: TDateTime) : 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` ([506](#)), `WeeksBetween` ([536](#)), `DaysBetween` ([468](#)), `HoursBetween` ([483](#)), `MinutesBetween` ([504](#)), `SecondsBetween` ([521](#)), `MillisecondsBetween` ([500](#)), `YearSpan` ([549](#))

Listing: `./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.

9.4.163 YearSpan

Synopsis: Calculate the approximate number of years between two `DateTime` 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` (507), `WeekSpan` (538), `DaySpan` (471), `HourSpan` (484), `MinuteSpan` (505), `SecondSpan` (522), `MilliSecondSpan` (501), `YearsBetween` (548)

Listing: `./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.

9.4.164 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 (528) -1`.

See also: `Today (528)`, `Tomorrow (528)`

Listing: `./datutex/ex18.pp`

Program `Example18;`

{ This program demonstrates the Yesterday function }

Uses `SysUtils , DateUtils ;`

Begin

`WriteLn (FormatDateTime('"Today is " dd mmm yyyy ', Today));`

`WriteLn (FormatDateTime('"Yesterday was " dd mmm yyyy ', Yesterday));`

End.

Chapter 10

Reference for unit 'Dos'

10.1 Used units

Table 10.1: Used units by unit 'Dos'

Name	Page
BaseUnix	100
System	1118

10.2 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 ([1360](#)) unit instead.

10.3 System information

Functions for retrieving and setting general system information such as date and time.

Table 10.2:

Name	Description
DosVersion (1)	Get OS version
GetCBreak (565)	Get setting of control-break handling flag
GetDate (565)	Get system date
GetIntVec (568)	Get interrupt vector status
GetTime (569)	Get system time
GetVerify (570)	Get verify flag
Intr (570)	Execute an interrupt
Keep (570)	Keep process in memory and exit
MSDos (571)	Execute MS-dos function call
PackTime (571)	Pack time for file time
SetCBreak (572)	Set control-break handling flag
SetDate (572)	Set system date
SetIntVec (573)	Set interrupt vectors
SetTime (574)	Set system time
SetVerify (574)	Set verify flag
SwapVectors (574)	Swap interrupt vectors
UnPackTime (575)	Unpack file time

10.4 Process handling

Functions to handle process information and starting new processes.

Table 10.3:

Name	Description
DosExitCode (1)	Exit code of last executed program
EnvCount (560)	Return number of environment variables
EnvStr (561)	Return environment string pair
Exec (561)	Execute program
GetEnv (566)	Return specified environment string

10.5 Directory and disk handling

Routines to handle disk information.

Table 10.4:

Name	Description
AddDisk (557)	Add disk to list of disks (UNIX only)
DiskFree (558)	Return size of free disk space
DiskSize (558)	Return total disk size

10.6 File handling

Routines to handle files on disk.

Table 10.5:

Name	Description
FExpand (562)	Expand filename to full path
FindClose (562)	Close finfirst/findnext session
FindFirst (562)	Start find of file
FindNext (563)	Find next file
FSearch (563)	Search for file in a path
FSplit (564)	Split filename in parts
GetFAttr (566)	Return file attributes
GetFTime (567)	Return file time
GetLongName (568)	Convert short filename to long filename (DOS only)
GetShortName (569)	Convert long filename to short filename (DOS only)
SetFAttr (572)	Set file attributes
SetFTime (573)	Set file time

10.7 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 10.6: Possible mode constants

Constant	Description	Value
<code>fmclosed</code>	File is closed	<code>\$D7B0</code>
<code>fminput</code>	File is read only	<code>\$D7B1</code>
<code>fmoutput</code>	File is write only	<code>\$D7B2</code>
<code>fminout</code>	File is read and write	<code>\$D7B3</code>

10.8 File attributes

The File Attribute constants are used in `FindFirst` (562), `FindNext` (563) to determine what type of special file to search for in addition to normal files. These flags are also used in the `SetFAttr` (572) and `GetFAttr` (566) routines to set and retrieve attributes of files. For their definitions consult `fileattributes` (553).

Table 10.7: 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

10.9 Constants, types and variables

10.9.1 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

Volumd ID file attribute

10.9.2 Types

ComStr = string

Command-line string type

DateTime = packed record

Year : Word;

Month : Word;

Day : Word;

Hour : Word;

Min : Word;

Sec : Word;

end

The `DateTime` type is used in `PackTime` (571) and `UnPackTime` (575) for setting/reading file times with `GetFTime` (567) and `SetFTime` (573).

```
DirStr = string
```

Full directory string type.

```
ExtStr = string
```

Filename extension string type.

```
NameStr = string
```

Full filename string type.

```
PathStr = string
```

Full File path string type.

```
Registers = packed record
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.

```
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` (562) call and can be used in subsequent `FindNext` (563) 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)

10.9.3 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 10.8: 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.

10.10 Procedures and functions

10.10.1 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` (558) and `DiskSize` (558) calls. The `DiskFree` (558) and `DiskSize` (558) 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` (558), `DiskSize` (558)

10.10.2 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` (557) 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` (558), `AddDisk` (557)

Listing: `./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.

```

10.10.3 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` (557) procedure. These settings can be coded in `dos.pp`, in the initialization part.

For an example, see `DiskFree` (558).

Errors: -1 when a failure occurs, or an invalid drive number is given.

See also: `DiskFree` (558), `AddDisk` (557)

10.10.4 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` (561)

Listing: `./dosex/ex5.pp`

```
Program Example5;
uses Dos;

{ Program to demonstrate the Exec and DosExitCode function. }

begin
  {$IFDEF Unix}
    WriteLn( 'Executing /bin/ls -la ' );
    Exec( '/bin/ls', '-la' );
  {$ELSE}
    WriteLn( 'Executing Dir ' );
    Exec( GetEnv( 'COMSPEC' ), '/C dir ' );
  {$ENDIF}
  WriteLn( 'Program returned with ExitCode ', Lo(DosExitCode) );
end.
```

10.10.5 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: ./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.

```

10.10.6 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 ([575](#)), PackTime ([571](#)), UnpackTime ([575](#)), GetTime ([569](#)), SetTime ([574](#))

10.10.7 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` (561), `GetEnv` (566)

10.10.8 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` (560), `GetEnv` (566)

Listing: `./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.

10.10.9 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` (1)

Errors: Errors are reported in `DosError`.

See also: `DosExitCode` (1)

10.10.10 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` ([564](#))

Listing: `./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.

```

10.10.11 FindClose

Synopsis: Dispose resources allocated by a `FindFirst` ([562](#))/`FindNext` ([563](#)) 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` ([562](#)) or `FindNext` ([563](#)) 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` ([562](#)), `FindNext` ([563](#))

10.10.12 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 os/2, you cannot issue two different `FindFirst` calls. That is, you must close any previous search operation with `FindClose` (562) 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` (563), `FindClose` (562)

Listing: `./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.

```

10.10.13 FindNext

Synopsis: Find next matching file after `FindFirst` (562)

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` (562).

Errors: `DosError` is used to report errors.

See also: `FindFirst` (562), `FindClose` (562)

10.10.14 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` ([562](#))

Listing: `./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.

```

10.10.15 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` ([563](#))

Listing: `./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.

```

10.10.16 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` ([572](#))

10.10.17 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` ([569](#)), `SetDate` ([572](#))

Listing: `./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.

```

10.10.18 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` ([560](#)), `EnvStr` ([561](#))

Listing: `./dosex/ex14.pp`

```

Program Example14;
uses Dos;

{ Program to demonstrate the GetEnv function. }

begin
  WriteLn('Current PATH is ',GetEnv('PATH'));
end .

```

10.10.19 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` ([572](#))

Listing: ./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.

```

10.10.20 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` ([573](#)), `PackTime` ([571](#)), `UnPackTime` ([575](#))

Listing: ./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.

```

10.10.21 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` ([573](#))

10.10.22 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` ([569](#))

10.10.23 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` ([569](#))

10.10.24 `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` ([568](#))

10.10.25 `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` ([565](#)), `SetTime` ([574](#))

Listing: `./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.

```

10.10.26 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 os/2 applications running under vanilla DOS), `Verify` is always `True`.

Errors: None.

See also: `SetVerify` ([574](#))

10.10.27 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` ([571](#))

10.10.28 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.

10.10.29 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` ([570](#))

10.10.30 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` ([573](#)), `FindFirst` ([562](#)), `FindNext` ([563](#)), `UnPackTime` ([575](#))

Listing: `./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.

```

10.10.31 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` ([565](#))

10.10.32 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` ([565](#)), `SetTime` ([574](#))

10.10.33 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` ([566](#))

10.10.34 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` ([567](#)), `PackTime` ([571](#)), `UnPackTime` ([575](#))

10.10.35 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` ([568](#))

10.10.36 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` ([569](#)), `SetDate` ([572](#))

10.10.37 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 os/2 applications running under vanilla dos), `Verify` is always `True`.

Errors: None.

See also: `SetVerify` ([574](#))

10.10.38 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` ([561](#)), `SetIntVec` ([573](#))

10.10.39 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` (560), `PackTime` (571), `UnpackTime` (575), `GetTime` (569), `SetTime` (574)

10.10.40 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` (571).

Errors: None.

See also: `GetFTime` (567), `FindFirst` (562), `FindNext` (563), `PackTime` (571)

10.10.41 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` (571), `UnpackTime` (575), `GetTime` (569), `SetTime` (574)

Chapter 11

Reference for unit 'dxeload'

11.1 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.

11.2 Procedures and functions

11.2.1 `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` ([580](#)) unit in the RTL.

Errors: If an error occurs during the load or relocations, `Nil` is returned.

Chapter 12

Reference for unit 'dynlibs'

12.1 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.

12.2 Constants, types and variables

12.2.1 Constants

`NilHandle = (0)`

Correctly typed Nil handle - returned on error by `LoadLibrary` ([579](#))

`SharedSuffix = 'so'`

`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.

12.2.2 Types

`HModule = TLibHandle`

Alias for `TLibHandle` ([577](#)) type.

`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.

`TOrdinalEntry = SizeUInt`

`TOrdinalEntry` is mainly used on Windows and OS/2 operating systems to retrieve the address of a procedure using the index (ordinal) of the procedure. On these operating systems, entry points can be loaded using name or ordinal value.

12.3 Procedures and functions

12.3.1 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` (579), and is provided for compatibility with Delphi.

See also: `UnloadLibrary` (579)

12.3.2 GetLoadErrorStr

Synopsis: Return a textual description of the last load error

Declaration: `function GetLoadErrorStr : string`

Visibility: default

Description: `GetLoadErrorStr` returns a textual description of the last library loading or unloading error. No other system calls may be made between the load call and the call of `GetLoadErrorStr`

See also: `LoadLibrary` (579)

12.3.3 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 `GetProcedureAddress` (578), and is provided for compatibility with Delphi.

See also: `GetProcedureAddress` (578)

12.3.4 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: `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` (579), `UnLoadLibrary` (579)

12.3.5 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` (577) 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` (577) is returned.

See also: `UnLoadLibrary` (579), `GetProcAddress` (578)

12.3.6 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` (579) with library name `Name`. After that function has returned, the FPU control word is saved again. (only on Intel i386 CPUS).

See also: `LoadLibrary` (579)

12.3.7 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 succesful, `False` otherwisa.

Errors: On error, `False` is returned.

See also: `LoadLibrary` (579), `GetProcAddress` (578)

Chapter 13

Reference for unit 'emu387'

13.1 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.

13.2 Procedures and functions

13.2.1 `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 14

Reference for unit 'exeinfo'

14.1 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` ([745](#)) and `Infodwrf` ([773](#)) unit to examine the binary for debug info.

14.2 Constants, types and variables

14.2.1 Types

```
TExeFile = record
  f : File;
  size : Int64;
  isopen : Boolean;
  nsects : LongInt;
  sechdrofs : ptruint;
  secstrofs : ptruint;
  processaddress : ptruint;
  FunctionRelative : Boolean;
  ImgOffset : ptruint;
  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.

14.3 Procedures and functions

14.3.1 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` (582), and represented by `e`.

The function returns `True` if the file was closed succesfully, or `False` if something went wrong.

Errors: In case of an error, `False` is returned.

See also: `OpenExeFile` (582)

14.3.2 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` (582) (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` (582)

14.3.3 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 succesful, it returns `True` and returns the filename in `FileName` and the base address at which it is loaded in `BaseAddr`.

14.3.4 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` ([582](#)), `CloseExeFile` ([582](#)), `ReadDebugLink` ([583](#))

14.3.5 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` ([582](#)), `CloseExeFile` ([582](#))

Chapter 15

Reference for unit 'getopts'

15.1 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 you program, and the unit will then parse the command-line options for you, and inform you of any errors.

15.2 Constants, types and variables

15.2.1 Constants

`EndOfOptions = #255`

Returned by `getopt` ([586](#)), `getlongopts` ([586](#)) 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.

15.2.2 Types

`Orderings = (require_order, permute, return_in_order)`

Table 15.1: 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` (585) record.

```
TOption = record
  Name : string;
  Has_arg : Integer;
  Flag : PChar;
  Value : Char;
end
```

The `TOption` type is used to communicate the long options to `GetLongOpts` (586). 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.

15.2.3 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.

15.3 Procedures and functions

15.3.1 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.

`ShortOptions` is a string containing all possible one-letter options. (see [Getopt \(586\)](#) 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`, if the option optionally has an argument, set `Has_arg` to `Optional_argument`. 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 \(586\)](#).

See also: [Getopt \(586\)](#)

15.3.2 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 \(586\)](#)

Listing: ./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',@theo[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 : ',theo[optionindex].name);
                if theopts[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.
end.

```

Chapter 16

Reference for unit 'go32'

16.1 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

16.2 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` ([612](#))
- %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` (612), `free_rm_callback` (607), `lock_code` (620), `lock_data` (620)

16.3 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` (630) and it's linear address via `transfer_buffer` (630). Another way is to allocate a completely new dos memory area via the `global_dos_alloc` (617) function for your use and supply its real mode address.

See also: `tb_size` (630), `transfer_buffer` (630), `global_dos_alloc` (617), `global_dos_free` (618), `realintr` (623)

Listing: `./go32ex/softint.pp`

```

uses
    go32;

var
    r : trealregs;

begin
    r.ah := $30;
    r.al := $01;
    realintr($21, r);
    Writeln('DOS v', r.al, '.', r.ah, ' detected');
end.
```

Listing: `./go32ex/rmpmint.pp`

```

uses
    crt,
    go32;

var
    r : trealregs;
    axreg : Word;

    oldint21h : tseginfo;
```

```

        newint21h : tseginfo;
procedure int21h_handler; assembler;
asm
        cmpw $0x3001, %ax
        jne .LCallOld
        movw $0x3112, %ax
        iret

.LCallOld:
        ljmp %cs:oldint21h
end;

procedure resume;
begin
        Writeln;
        Write('— press any key to resume —'); readkey;
        gotoxy(1, wherey); clreol;
end;

begin
        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;
WriteIn('DOS v', lo(axreg), '.', hi(axreg), ' detected');
WriteIn;
WriteIn('Now you can see that there''s a distinction between ',
        'the two ways of calling interrupts...');
set_pm_interrupt($21, oldint21h);
end.

```

16.4 Software interrupts

Ordinarily, a handler installed with `set_pm_interrupt` (627) only services software interrupts that are executed in protected mode; real mode software interrupts can be redirected by `set_rm_interrupt` (628).

See also: `set_rm_interrupt` (628), `get_rm_interrupt` (615), `set_pm_interrupt` (627), `get_pm_interrupt` (611), `lock_data` (620), `lock_code` (620), `enable` (606), `disable` (604), `outportb` (621)

16.5 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: ./go32ex/keyclick.pp

```

{$ASMMODE ATT}
{$MODE FPC}

uses
    crt ,
    go32;

const
    kbdint = $9;

```

```

var
    oldint9_handler : tseginfo;
    newint9_handler : tseginfo;

    clickproc : pointer;
    backupDS : Word; external name '___v2prt0_ds_alias';

procedure int9_handler; assembler;
asm
    cli
    pushl %ds
    pushl %es
    pushl %fs
    pushl %gs
    pushal
    movw %cs:backupDS, %ax
    movw %ax, %ds
    movw %ax, %es
    movw dosmemselector, %ax
    movw %ax, %fs
    call *clickproc
    popal
    popl %gs
    popl %fs
    popl %es
    popl %ds
    ljmp %cs:oldint9_handler
end;
procedure int9_dummy; begin end;

procedure clicker;
begin
    sound(500); delay(10); nosound;
end;
procedure clicker_dummy; begin end;

procedure install_click;
begin
    clickproc := @clicker;
    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));
    newint9_handler.offset := @int9_handler;
    newint9_handler.segment := get_cs;
    get_pm_interrupt(kbdint, oldint9_handler);
    set_pm_interrupt(kbdint, newint9_handler);
end;

procedure remove_click;
begin
    set_pm_interrupt(kbdint, oldint9_handler);
    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.

```

16.6 Disabling interrupts

The GO32 unit provides the two procedures `disable()` and `enable()` to disable and enable all interrupts.

16.7 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.

16.8 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.

16.9 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.

16.10 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.

16.11 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` (607), `get_ds` (608), `get_ss` (617)

16.12 I/O port access

The I/O port access is done via the various `inportb` (619), `outportb` (621) 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` (621), `inportb` (619)

16.13 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` (597), `dosmemput` (597), `dosmemmove` (597), `dosmemfillchar` (596), `dosmemfillword` (597), `seg_move` (625), `seg_fillchar` (624), `seg_fillword` (625)

16.14 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` (607), `get_ds` (608), `get_ss` (617), `allocate_ldt_descriptors` (600), `free_ldt_descriptor` (606), `segment_to_descriptor` (624), `get_next_selector_increment_value` (610), `get_segment_base_address` (616), `set_segment_base_address` (628), `set_segment_limit` (629), `create_code_segment_alias_descriptor` (604)

16.15 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.

16.16 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.

16.17 Constants, types and variables

16.17.1 Constants

`auxcarryflag = $010`

Check for auxiliary carry flag in `trealregs` (600)

`carryflag = $001`

Check for carry flag in `trealregs` (600)

`directionflag = $400`

Check for direction flag in `trealregs` (600)

`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 \(617\)](#).

```
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` (617).

```
interruptflag = $200
```

Check for interrupt flag in `trealregs` (600)

```
overflowflag = $800
```

Check for overflow flag in `trealregs` (600)

```
parityflag = $004
```

Check for parity flag in `trealregs` (600)

```
rm_dpml = 4
```

`get_run_mode` (615) return value: DPMI (e.g. dos box or 386Max)

```
rm_raw = 1
```

`get_run_mode` (615) return value: raw (without HIMEM)

```
rm_unknown = 0
```

`get_run_mode` (615) return value: Unknown runmode

```
rm_vcpi = 3
```

`get_run_mode` (615) return value: VCPI (with HIMEM and EMM386)

```
rm_xms = 2
```

`get_run_mode` (615) return value: XMS (with HIMEM, without EMM386)

```
signflag = $080
```

Check for sign flag in `trealregs` (600)

```
trapflag = $100
```

Check for trap flag in `trealregs` (600)

```
zeroflag = $040
```

Check for zero flag in `trealregs` (600)

16.17.2 Types

registers = trealregs

Alias for trealregs (600)

```
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` (608). 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.

```
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.

```
trealregs = record
end
```

The `trealregs` type contains the data structure to pass register values to a interrupt handler or real mode callback.

```
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

16.17.3 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 DPPI interrupt call. Any nonzero value must be treated as a critical failure.

16.18 Procedures and functions

16.18.1 `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` (610) function.

Errors: Check the `int31error` (600) variable.

See also: `free_ldt_descriptor` (606), `get_next_selector_increment_value` (610), `segment_to_descriptor` (624), `create_code_segment_alias_descriptor` (604), `set_segment_limit` (629), `set_segment_base_address` (628)

Listing: `./go32ex/seldes.pp`

```

{$mode delphi}
uses
    crt ,
    go32;

const
    maxx = 80;
    maxy = 25;
    bytespercell = 2;
    screensize = maxx * maxy * bytespercell;

    linB8000 = $B800 * 16;

type
    string80 = string[80];

var
    text_save : array[0..screensize-1] of byte;
    text_oldx , text_oldy : Word;

    text_sel : Word;

procedure status(s : string80);
begin
    gotoxy(1, 1); clreol; write(s); readkey;
end;

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;

function makechar(ch : char; color : byte) : Word;
begin
    result := byte(ch) or (color shl 8);
end;

begin
    seg_move(dosmemselector, linB8000, get_ds, longint(@text_save),
        screensize);
    text_oldx := wherex; text_oldy := wherey;
    seg_fillword(dosmemselector, linB8000, screensize div 2,
        makechar(' ', Black or (Black shl 4)));
    status('Creating selector ''text_sel'' to a part of ' +
        'text screen memory');
    text_sel := allocate_ldt_descriptors(1);
    set_segment_base_address(text_sel,
        linB8000 + bytespercell * maxx * 1);
    set_segment_limit(text_sel, screensize - 1 - bytespercell *
        maxx * 3);

```

```

selinfo(text_sel);

status('and clearing entire memory selected by ''text_sel'' +
      ' descriptor');
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_segment_base_address(text_sel,
                         get_segment_base_address(text_sel) + bytespercell * maxx);
set_segment_limit(text_sel,
                  get_segment_limit(text_sel) - bytespercell * maxx * 2);
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');
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');
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_ldt_descriptor(text_sel);
seg_move(get_ds, longint(@text_save), dosmemselector,
         linB8000, screensize);
gotoxy(text_oldx, text_oldy);
end.

```

16.18.2 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` (600) variable.

See also: `free_memory_block` (606)

16.18.3 `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` (600) variable.

See also: `tb_size` (630), `transfer_buffer` (630), `copytodos` (603)

16.18.4 `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` (600) variable.

See also: `tb_size` (630), `transfer_buffer` (630), `copyfromdos` (603)

16.18.5 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:

segDescriptor.

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` (600) variable.

See also: `allocate_ldt_descriptors` (600), `set_segment_limit` (629), `set_segment_base_address` (628)

16.18.6 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` (606)

16.18.7 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_dosmempout` (605), `dpmi_dosmemget` (605), `dpmi_dosmemmove` (605), `dpmi_dosmemfillword` (605)

16.18.8 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` (605), `dpmi_dosmemget` (605), `dpmi_dosmemfillchar` (604), `dpmi_dosmemmove` (605)

16.18.9 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` (605), `dpmi_dosmemmove` (605), `dpmi_dosmemfillchar` (604), `dpmi_dosmemfillword` (605)

16.18.10 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` (605), `dpmi_dosmemget` (605), `dpmi_dosmemfillchar` (604), `dpmi_dosmemfillword` (605)

16.18.11 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` (605), `dpmi_dosmemmove` (605), `dpmi_dosmemfillchar` (604), `dpmi_dosmemfillword` (605)

16.18.12 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` (604)

16.18.13 free_ldt_descriptor

Synopsis: Free a descriptor

Declaration: `function free_ldt_descriptor(d: Word) : Boolean`

Visibility: default

Description: Frees a previously allocated descriptor.

Parameters:

desThe 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` (600) 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` (600).

Errors: Check the `int31error` (600) variable.

See also: `allocate_ldt_descriptors` (600), `get_next_selector_increment_value` (610)

16.18.14 free_linear_addr_mapping

Synopsis: ? No description available

Declaration: `function free_linear_addr_mapping(linear_addr: dword) : Boolean`

Visibility: default

16.18.15 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:

blockhandlethe 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` (602) . This function doesn't free any descriptors mapped to this block, it's the application's responsibility.

Errors: Check `int31error` (600) variable.

See also: `allocate_memory_block` (602)

16.18.16 `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` (612) function.

Parameters:

intaddr real mode address buffer returned by `get_rm_callback` (612) .

Return values: `True` if successful, `False` if not

For an example, see `get_rm_callback` (612).

Errors: Check the `int31error` (600) variable.

See also: `set_rm_interrupt` (628), `get_rm_callback` (612)

16.18.17 `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` (627).

Errors: None.

See also: `get_ds` (608), `get_ss` (617)

16.18.18 `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:

`dselector` to descriptor.

Return value: Access rights bit field.

Errors: Check the `int31error` (600) variable.

See also: `set_descriptor_access_right` (626)

16.18.19 `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` (599)

16.18.20 `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` (607), `get_ss` (617)

16.18.21 `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` (626), `get_pm_exception_handler` (611)

16.18.22 get_linear_addr

Synopsis: Convert physical to linear address

Declaration: `function get_linear_addr(phys_addr: LongInt; size: LongInt) : LongInt`

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` (600) variable.

See also: `allocate_ldt_descriptors` (600), `set_segment_limit` (629), `set_segment_base_address` (628)

16.18.23 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` (611) function.

Errors: Check the `int31error` (600) variable.

See also: `get_page_size` (611)

Listing: `./go32ex/meminfo.pp`

```
uses
    go32;

var
    meminfo : tmeminfo;

begin
    get_meminfo(meminfo);
    if (int31error <> 0) then begin
```

```

        Writeln('Error getting DPMI memory information... Halting');
        Writeln('DPMI 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.

```

16.18.24 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 (600).

Return value: Selector increment value.

Remark: Notes: Because allocate_ldt_descriptors (600) 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 (600) variable.

See also: allocate_ldt_descriptors (600), free_ldt_descriptor (606)

16.18.25 get_page_attributes

Synopsis: ? No description available

Declaration: `function get_page_attributes(handle: dword; offset: dword;
pagecount: dword; buf: pointer) : Boolean`

Visibility: default

16.18.26 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` (609).

Errors: Check the `int31error` (600) variable.

See also: `get_meminfo` (609)

16.18.27 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` (608), `set_pm_exception_handler` (627)

16.18.28 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` (627).

Errors: Check the `int31error` (600) variable.

See also: `set_pm_interrupt` (627), `set_rm_interrupt` (628), `get_rm_interrupt` (615)

16.18.29 `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.

regs 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` (600) variable.

See also: `free_rm_callback` (607)

Listing: `./go32ex/callback.pp`

```
{ $ASMMODE ATT }
{ $MODE FPC }

uses
    crt ,
    go32;

const
    mouseint = $33;

var
    mouse_regs      : trealregs; external name '___v2prt0_rmcb_regs';
    mouse_seginfo   : tseginfo;

var
    mouse_numbuttons : longint;

    mouse_action     : word;
    mouse_x, mouse_y : Word;
    mouse_b          : Word;

    userproc_installed : Longbool;
    userproc_length   : Longint;
```

```

        userproc_proc : pointer;

procedure callback_handler; assembler;
asm
    pushw %ds
    pushl %eax
    movw %es, %ax
    movw %ax, %ds

    cmpl $1, USERPROC_INSTALLED
    jne .LNoCallback
    pushal
    movw DOSmemSELECTOR, %ax
    movw %ax, %fs
    call *USERPROC_PROC
    popal
.LNoCallback:

    popl %eax
    popw %ds

    pushl %eax
    movl (%esi), %eax
    movl %eax, %es: 42(%edi)
    addw $4, %es:46(%edi)
    popl %eax
    iret
end;
procedure mouse_dummy; begin end;

procedure textuserproc;
begin
    mouse_b := mouse_regs.bx;
    mouse_x := (mouse_regs.cx shr 3) + 1;
    mouse_y := (mouse_regs.dx shr 3) + 1;
end;

procedure install_mouse(userproc : pointer; userproclen : longint);
var r : treatregs;
begin
    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;
    if (r.bx = $ffff) then mouse_numbuttons := 2
    else mouse_numbuttons := r.bx;
    Writeln(mouse_numbuttons, ' button Microsoft compatible mouse ',
        ' found. ');
    if (userproc <> nil) then begin
        userproc_proc := userproc;
        userproc_installed := true;
        userproc_length := userproclen;
        lock_code(userproc_proc, userproc_length);
    end else begin
        userproc_proc := nil;

```

```

        userproc_length := 0;
        userproc_installed := false;
    end;
    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_seginfo, sizeof(mouse_seginfo));
    lock_code(@callback_handler,
        longint(@mouse_dummy) - longint(@callback_handler));
    get_rm_callback(@callback_handler, mouse_regs, mouse_seginfo);
    r.eax := $0c; r.ecx := $7f;
    r.edx := longint(mouse_seginfo.offset);
    r.es := mouse_seginfo.segment;
    realintr(mouseint, r);
    r.eax := $01;
    realintr(mouseint, r);
end;

procedure remove_mouse;
var
    r : trealregs;
begin
    r.eax := $02; realintr(mouseint, r);
    r.eax := $0c; r.ecx := 0; r.edx := 0; r.es := 0;
    realintr(mouseint, r);
    free_rm_callback(mouse_seginfo);
    if (userproc_installed) then begin
        unlock_code(userproc_proc, userproc_length);
        userproc_proc := nil;
        userproc_length := 0;
        userproc_installed := false;
    end;
    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

```

```

        gotoxy(1, wherey);
        write('MouseX : ', mouse_x:2, ' MouseY : ', mouse_y:2,
              ' Buttons : ', mouse_b:2);
    end;
    remove_mouse;
end.

```

16.18.30 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` (600) variable.

See also: `set_rm_interrupt` (628), `set_pm_interrupt` (627), `get_pm_interrupt` (611)

16.18.31 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` (615)

Listing: `./go32ex/getrunmd.pp`

```

uses
    go32;

begin
    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_vcp   :
        WriteLn('VCPI server detected. You''re using HIMEM and ',
                'EMM386');
rm_dp    :
        WriteLn('DPMI detected. You''re using a DPMI host like ',
                'a windows DOS box or CWSDPMI');
    end;
end.

```

16.18.32 get_segment_base_address

Synopsis: Return base address from descriptor table

Declaration: `function get_segment_base_address(d: Word) : LongInt`

Visibility: default

Description: Returns the 32-bit linear base address from the descriptor table for the specified segment.

Parameters:

dselector of the descriptor you want the base address of.

Return values: Linear base address of specified descriptor.

For an example, see `allocate_ldt_descriptors` (600).

Errors: Check the `int31error` (600) variable.

See also: `allocate_ldt_descriptors` (600), `set_segment_base_address` (628), `allocate_ldt_descriptors` (600), `set_segment_limit` (629), `get_segment_limit` (616)

16.18.33 get_segment_limit

Synopsis: Return segment limit from descriptor

Declaration: `function get_segment_limit(d: Word) : LongInt`

Visibility: default

Description: Returns a descriptors segment limit.

Parameters:

dselector.

Return value: Limit of the descriptor in bytes.

Errors: Returns zero if descriptor is invalid.

See also: `allocate_ldt_descriptors` (600), `set_segment_limit` (629), `set_segment_base_address` (628), `get_segment_base_address` (616)

16.18.34 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` (608), `get_cs` (607)

16.18.35 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:

bytessize 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 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 an unique selector. The returned selector should only be freed by a `global_dos_free` (618) call.

Errors: Check the `int31error` (600) variable.

See also: `global_dos_free` (618)

Listing: `./go32ex/buffer.pp`

```

uses
    go32;

procedure dosalloc(var selector : word;
                   var segment : word; size : longint);
var
    res : longint;
begin
    res := global_dos_alloc(size);
    selector := word(res);
    segment := word(res shr 16);
end;

procedure dosfree(selector : word);
begin
```

```

    global_dos_free(selector);
end;

type
    VBEInfoBuf = packed record
        Signature : array[0..3] of char;
        Version : Word;
        reserved : array[0..505] of byte;
    end;

var
    selector ,
    segment : Word;

    r : trealregs;
    infobuf : VBEInfoBuf;

begin
    fillchar(r, sizeof(r), 0);
    fillchar(infobuf, sizeof(VBEInfoBuf), 0);
    dosalloc(selector, segment, sizeof(VBEInfoBuf));
    if (int31error <> 0) then begin
        WriteLn('Error while allocating real mode memory, halting');
        halt;
    end;
    infobuf.Signature := 'VBE2';
    dosmempu(segment, 0, infobuf, sizeof(infobuf));
    r.ax := $4f00; r.es := segment;
    realintr($10, r);
    dosmemget(segment, 0, infobuf, sizeof(infobuf));
    dosfree(selector);
    if (r.ax <> $4f) then begin
        WriteLn('VBE BIOS extension not available, function call ',
            'failed');
        halt;
    end;
    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.

```

16.18.36 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` (617).

For an example, see `global_dos_alloc` (617).

Errors: Check the `int31error` (600) variable.

See also: `global_dos_alloc` (617)

16.18.37 `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` (621), `inportw` (619), `inportl` (619)

16.18.38 `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` (621), `inportb` (619), `inportw` (619)

16.18.39 `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 \(622\)](#), [inportb \(619\)](#), [inportl \(619\)](#)

16.18.40 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 \(612\)](#).

Errors: Check the [int3error \(600\)](#) variable.

See also: [lock_linear_region \(621\)](#), [lock_data \(620\)](#), [unlock_linear_region \(631\)](#), [unlock_data \(631\)](#), [unlock_code \(630\)](#)

16.18.41 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 \(612\)](#).

Errors: Check the [int3error \(600\)](#) variable.

See also: [lock_linear_region \(621\)](#), [lock_code \(620\)](#), [unlock_linear_region \(631\)](#), [unlock_data \(631\)](#), [unlock_code \(630\)](#)

16.18.42 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` (600) variable.

See also: `lock_data` (620), `lock_code` (620), `unlock_linear_region` (631), `unlock_data` (631), `unlock_code` (630)

16.18.43 map_device_in_memory_block

Synopsis: Map a device into program's memory space

Declaration: `function map_device_in_memory_block(handle: LongInt;offset: LongInt;
pagecount: LongInt;device: LongInt)
: 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.

16.18.44 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` (619), `outportl` (622), `outportw` (622)

Listing: `./go32ex/outport.pp`

uses

```
crt ,
go32;
```

begin

```
outportb($61, $ff);
delay(50);
outportb($61, $0);
```

end.

16.18.45 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` ([621](#)).

Errors: None.

See also: `inportl` ([619](#)), `outportw` ([622](#)), `outportb` ([621](#))

16.18.46 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` ([621](#)).

Errors: None.

See also: `inportw` ([619](#)), `outportl` ([622](#)), `outportb` ([621](#))

16.18.47 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 `intnr`. The real mode handler must return by executing an `IRET`.

Errors: Check the `int31error` (600) variable.

Listing: `./go32ex/flags.pp`

```

uses
    go32;

var
    r : trealregs;

begin
    r.ax := $5300;
    r.bx := 0;
    realintr($15, r);
    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.

```

16.18.48 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.

16.18.49 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` (600) and change the base address as necessary.

For an example, see `seg_fillchar` (624).

Errors: Check the `int31error` (600) variable.

See also: `allocate_ldt_descriptors` (600), `free_ldt_descriptor` (606), `set_segment_base_address` (628)

16.18.50 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` (625), `seg_fillword` (625), `dosmemfillchar` (596), `dosmemfillword` (597), `dosmemget` (597), `dosmemput` (597), `dosmemmove` (597)

Listing: `./go32ex/vgasel.pp`

```

uses
    go32;

var
    vgasel : Word;
    r : trealregs;

begin
    r.eax := $13; realintr($10, r);
    vgasel := segment_to_descriptor($A000);
    seg_fillchar(vgasel, 0, 64000, #15);
    readln;
    r.eax := $3; realintr($10, r);

end.

```

16.18.51 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` ([600](#)).

Errors: None.

See also: `seg_move` ([625](#)), `seg_fillchar` ([624](#)), `dosmemfillchar` ([596](#)), `dosmemfillword` ([597](#)), `dosmemget` ([597](#)), `dosmemput` ([597](#)), `dosmemmove` ([597](#))

16.18.52 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.

sourcesource offset.

dsegdestination selector.

destdestination offset.

countsize 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` (600).

Errors: None.

See also: `seg_fillchar` (624), `seg_fillword` (625), `dosmemfillchar` (596), `dosmemfillword` (597), `dosmemget` (597), `dosmemput` (597), `dosmemmove` (597)

16.18.53 `set_descriptor_access_right`

Synopsis: Set access rights to memory descriptor

Declaration: `function set_descriptor_access_right(d: Word;w: Word) : LongInt`

Visibility: default

Description: `set_descriptor_access_right` sets the access rights for descriptor `d` to `w`

16.18.54 `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` (608), `set_pm_exception_handler` (627)

16.18.55 `set_page_attributes`

Synopsis: ? No description available

Declaration: `function set_page_attributes(handle: dword;offset: dword;
pagecount: dword;buf: pointer) : Boolean`

Visibility: default

16.18.56 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` (626), `get_pm_exception_handler` (611)

16.18.57 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 `int3lerror` (600) variable.

See also: `get_pm_interrupt` (611), `set_rm_interrupt` (628), `get_rm_interrupt` (615)

Listing: `./go32ex/intpm.pp`

uses

`crt ,
go32;`

const

`int1c = $1c;`

var

`oldint1c : tseginfo;
newint1c : tseginfo;`

`int1c_counter : Longint;`

`int1c_ds : Word; external name '___v2prt0_ds_alias';`

procedure `int1c_handler`; **assembler**;

asm

`cli
pushw %ds
pushw %ax`

```

    movw %cs:int1c_ds, %ax
    movw %ax, %ds
    incl int1c_counter
    popw %ax
    popw %ds
    sti
    iret
end;

var i : Longint;

begin
    newint1c.offset := @int1c_handler;
    newint1c.segment := get_cs;
    get_pm_interrupt(int1c, oldint1c);
    Writeln('-- Press any key to exit --');
    set_pm_interrupt(int1c, newint1c);
    while (not keypressed) do begin
        gotoxy(1, wherey);
        write('Number of interrupts occurred : ', int1c_counter);
    end;
    set_pm_interrupt(int1c, oldint1c);
end.

```

16.18.58 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` (612).

Errors: Check the `int31error` (600) variable.

See also: `get_rm_interrupt` (615), `set_pm_interrupt` (627), `get_pm_interrupt` (611), `get_rm_callback` (612)

16.18.59 set_segment_base_address

Synopsis: Set descriptor's base address

Declaration: `function set_segment_base_address(d: Word; s: LongInt) : Boolean`

Visibility: default

Description: Sets the 32-bit linear base address of a descriptor.

Parameters:

dselector.

snew base address of the descriptor.

Errors: Check the `int31error` (600) variable.

See also: `allocate_ldt_descriptors` (600), `get_segment_base_address` (616), `allocate_ldt_descriptors` (600), `set_segment_limit` (629), `get_segment_base_address` (616), `get_segment_limit` (616)

16.18.60 `set_segment_limit`

Synopsis: Set descriptor limit

Declaration: `function set_segment_limit(d: Word; s: LongInt) : Boolean`

Visibility: default

Description: Sets the limit of a descriptor.

Parameters:

dselector.

snew 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` (600).

Errors: Check the `int31error` (600) variable.

See also: `allocate_ldt_descriptors` (600), `set_segment_base_address` (628), `get_segment_limit` (616), `set_segment_limit` (629)

16.18.61 `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` (630), `tb_segment` (629), `tb_size` (630)

16.18.62 `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` (630), `tb_offset` (629), `tb_size` (630)

16.18.63 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` (630), `copyfromdos` (603), `copytodos` (603)

16.18.64 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` (630)

16.18.65 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` (612).

Errors: Check the `int31error` (600) variable.

See also: `unlock_linear_region` (631), `unlock_data` (631), `lock_linear_region` (621), `lock_data` (620), `lock_code` (620)

16.18.66 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` (612).

Errors: Check the `int31error` (600) variable.

See also: `unlock_linear_region` (631), `unlock_code` (630), `lock_linear_region` (621), `lock_data` (620), `lock_code` (620)

16.18.67 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` (600) variable.

See also: `unlock_data` (631), `unlock_code` (630), `lock_linear_region` (621), `lock_data` (620), `lock_code` (620)

Chapter 17

Reference for unit 'gpm'

17.1 Used units

Table 17.1: Used units by unit 'gpm'

Name	Page
BaseUnix	100
System	1118

17.2 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.

17.3 Constants, types and variables

17.3.1 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 ([640](#)).

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.

17.3.2 Types

`Pgpmconnect = Pgpm_connect`

Pointer to `TGpmConnect` (635) record.

`Pgpmevent = Pgpm_event`

Pointer to TGpmEvent (635) record

```
Pgpmroi = Pgpm_roi
```

Pointer to TGpmRoi (635) record.

```
Pgpm_connect = ^TGpm_connect
```

Pointer to TGpm_Connect (636) record.

```
Pgpm_event = ^Tgpm_event
```

Pointer to TGpm_Event (636) record

```
Pgpm_roi = ^Tgpm_roi
```

Pointer to Tgpm_roi (636) record.

```
Tgpmconnect = Tgpm_connect
```

Alias for TGpm_Connect (636) record.

```
TGpmEtype = LongInt
```

Type for event type.

```
Tgpmevent = Tgpm_event
```

Alias for TGPM_EVent (636) 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 (636)Record

```
Tgpm_connect = record
  eventMask : Word;
  defaultMask : Word;
  minMod : Word;
  maxMod : Word;
  pid : LongInt;
  vc : LongInt;
end
```


GPM server connection information.

```
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.

```
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.

17.3.3 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.

17.4 Procedures and functions

17.4.1 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` (642), `Gpm_AnySingle` (637), `Gpm_StrictDouble` (642), `Gpm_StrictTriple` (642), `Gpm_AnyTriple` (637)

17.4.2 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` (642), `Gpm_AnyDouble` (637), `Gpm_StrictDouble` (642), `Gpm_StrictTriple` (642), `Gpm_AnyTriple` (637)

17.4.3 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` (642), `Gpm_AnyDouble` (637), `Gpm_StrictDouble` (642), `Gpm_StrictTriple` (642), `Gpm_AnySingle` (637)

17.4.4 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` (638).

Errors: None.

See also: `Gpm_Open` (640)

17.4.5 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` (638)

17.4.6 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` (638)

17.4.7 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 succes, and -1 on failue.

Errors: On error, -1 is returned.

See also: `Gpm_GetSnapshot` (640)

Listing: `./gpmex/gpmex.pp`

```

program gpmex;

{
  Example program to demonstrate the use of the gpm unit.
}

uses gpm;

var
  connect : TGPMConnect;
  event : tgpmevent;

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;
        Writeln;
      end;
    Until (Event.Buttons and gpm_b_right)<>0;
    gpm_close;
  end.

```

17.4.8 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` (638)

17.4.9 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` (641), `Gpm_PopRoi` (641), `Gpm_PushRoi` (641)

17.4.10 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` (635).

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` (638).

Errors: On Error, the return value is -1.

See also: `Gpm_Open` (640)

17.4.11 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` (641), `Gpm_LowerRoi` (640), `Gpm_PushRoi` (641)

17.4.12 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` (641), `Gpm_PopRoi` (641), `Gpm_LowerRoi` (640)

17.4.13 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` (641), `Gpm_PopRoi` (641), `Gpm_LowerRoi` (640)

17.4.14 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 \(638\)](#)

17.4.15 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 \(642\)](#), [Gpm_AnyTriple \(637\)](#), [Gpm_AnyDouble \(637\)](#), [Gpm_StrictTriple \(642\)](#), [Gpm_AnySingle \(637\)](#)

17.4.16 Gpm_StrictSingle

Synopsis: Check whether event contains only a single-click event.

Declaration: `function Gpm_StrictSingle(EventType: LongInt) : Boolean`

Visibility: default

Description: `Gpm_StrictSingle` returns `True` if `EventType` contains only a singleclick event, `False` otherwise.

Errors: None.

See also: [Gpm_AnyTriple \(637\)](#), [Gpm_StrictDouble \(642\)](#), [Gpm_AnyDouble \(637\)](#), [Gpm_StrictTriple \(642\)](#), [Gpm_AnySingle \(637\)](#)

17.4.17 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 \(637\)](#), [Gpm_StrictDouble \(642\)](#), [Gpm_AnyDouble \(637\)](#), [Gpm_StrictSingle \(642\)](#), [Gpm_AnySingle \(637\)](#)

Chapter 18

Reference for unit 'Graph'

18.1 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.

18.2 Categorized functions: Text and font handling

Functions to set texts on the screen.

Table 18.1:

Name	Description
<code>GetTextSettings</code> (683)	Get current text settings
<code>InstallUserFont</code> (686)	Install a new font
<code>OutText</code> (687)	Write text at current cursor position
<code>OutTextXY</code> (675)	Write text at coordinates X,Y
<code>RegisterBGIFont</code> (689)	Register a new font
<code>SetTextJustify</code> (692)	Set text justification
<code>SetTextStyle</code> (692)	Set text style
<code>SetUserCharSize</code> (693)	Set text size
<code>TextHeight</code> (694)	Calculate height of text
<code>TextWidth</code> (694)	Calculate width of text

18.3 Categorized functions: Filled drawings

Functions for drawing filled regions.

Table 18.2:

Name	Description
Bar3D (676)	Draw a filled 3D-style bar
Bar (676)	Draw a filled rectangle
FloodFill (678)	Fill starting from coordinate
FillEllipse (678)	Draw a filled ellipse
FillPoly (678)	Draw a filled polygone
GetFillPattern (680)	Get current fill pattern
GetFillSettings (680)	Get current fill settings
SetFillPattern (690)	Set current fill pattern
SetFillStyle (690)	Set current fill settings

18.4 Categorized functions: Drawing primitives

Functions for simple drawing.

Table 18.3:

Name	Description
Arc (676)	Draw an arc
Circle (673)	Draw a complete circle
DrawPoly (677)	Draw a polygone with N points
Ellipse (678)	Draw an ellipse
GetArcCoords (679)	Get arc coordinates
GetLineSettings (681)	Get current line drawing settings
Line (675)	Draw line between 2 points
LineRel (686)	Draw line relative to current position
LineTo (687)	Draw line from current position to absolute position
MoveRel (687)	Move cursor relative to current position
MoveTo (687)	Move cursor to absolute position
PieSlice (688)	Draw a pie slice
PutPixel (675)	Draw 1 pixel
Rectangle (688)	Draw a non-filled rectangle
Sector (689)	Draw a sector
SetLineStyle (691)	Set current line drawing style

18.5 Categorized functions: Color management

All functions related to color management.

Table 18.4:

Name	Description
GetBkColor (674)	Get current background color
GetColor (679)	Get current foreground color
GetDefaultPalette (679)	Get default palette entries
GetMaxColor (681)	Get maximum valid color
GetPaletteSize (683)	Get size of palette for current mode
GetPixel (674)	Get color of selected pixel
GetPalette (683)	Get palette entry
SetAllPalette (675)	Set all colors in palette
SetBkColor (675)	Set background color
SetColor (690)	Set foreground color
SetPalette (692)	Set palette entry
SetRGBPalette (676)	Set palette entry with RGB values

18.6 Categorized functions: Screen management

General drawing screen management functions.

Table 18.5:

Name	Description
ClearViewPort (674)	Clear the current viewport
GetImage (674)	Copy image from screen to memory
GetMaxX (682)	Get maximum X coordinate
GetMaxY (682)	Get maximum Y coordinate
GetX (684)	Get current X position
GetY (684)	Get current Y position
ImageSize (674)	Get size of selected image
GetViewSettings (683)	Get current viewport settings
PutImage (675)	Copy image from memory to screen
SetActivePage (675)	Set active video page
SetAspectRatio (689)	Set aspect ratio for drawing routines
SetViewPort (693)	Set current viewport
SetVisualPage (676)	Set visual page
SetWriteMode (694)	Set write mode for screen operations

18.7 Categorized functions: Initialization

Initialization of the graphics screen.

Table 18.6:

Name	Description
<code>ClearDevice</code> (677)	Empty the graphics screen
<code>CloseGraph</code> (677)	Finish drawing session, return to text mode
<code>DetectGraph</code> (677)	Detect graphical modes
<code>GetAspectRatio</code> (679)	Get aspect ratio of screen
<code>GetModeRange</code> (682)	Get range of valid modes for current driver
<code>GraphDefaults</code> (1)	Set defaults
<code>GetDriverName</code> (680)	Return name of graphical driver
<code>GetGraphMode</code> (681)	Return current or last used graphics mode
<code>GetMaxMode</code> (681)	Get maximum mode for current driver
<code>GetModeName</code> (682)	Get name of current mode
<code>GraphErrorMsg</code> (1)	String representation of graphical error
<code>GraphResult</code> (1)	Result of last drawing operation
<code>InitGraph</code> (685)	Initialize graphics drivers
<code>InstallUserDriver</code> (686)	Install a new driver
<code>RegisterBGIDriver</code> (688)	Register a new driver
<code>RestoreCRTMode</code> (689)	Go back to text mode
<code>SetGraphMode</code> (691)	Set graphical mode

18.8 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 `-Fl` 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.

18.9 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.

18.10 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: ./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: ./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: ./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.

```

18.11 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).

18.12 Constants, types and variables

18.12.1 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, $80, $81, $82, $83, $84, $85, $86, $87, $88, $89, $8A, $8B, $8C, $8D, $8E, $8F, $90, $91, $92, $93, $94, $95, $96, $97, $98, $99, $9A, $9B, $9C, $9D, $9E, $9F, $A0, $A1, $A2, $A3, $A4, $A5, $A6, $A7, $A8, $A9, $AA, $AB, $AC, $AD, $AE, $AF, $B0, $B1, $B2, $B3, $B4, $B5, $B6, $B7, $B8, $B9, $BA, $BB, $BC, $BD, $BE, $BF, $C0, $C1, $C2, $C3, $C4, $C5, $C6, $C7, $C8, $C9, $CA, $CB, $CC, $CD, $CE, $CF, $D0, $D1, $D2, $D3, $D4, $D5, $D6, $D7, $D8, $D9, $DA, $DB, $DC, $DD, $DE, $DF, $E0, $E1, $E2, $E3, $E4, $E5, $E6, $E7, $E8, $E9, $EA, $EB, $EC, $ED, $EE, $EF, $F0, $F1, $F2, $F3, $F4, $F5, $F6, $F7, $F8, $F9, $FA, $FB, $FC, $FD, $FE, $FF)`

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,

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 = (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),

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

18.12.2 Types

```
ArcCoordsType = record
  x : SmallInt;
  y : SmallInt;
  xstart : SmallInt;
  ystart : SmallInt;
  xend : SmallInt;
  yend : SmallInt;
end
```

Describe the last arc which was drawn on screen

```
CircleProc = procedure(X: SmallInt;Y: SmallInt;Radius: Word)
```

Standard circle drawing routine prototype.

```
clrviewproc = procedure
```

Standard clearviewport routine prototype

```
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.

```
FillSettingsType = record
    pattern : Word;
    color : Word;
end
```

Record describing fill mode

```
GetBkColorProc = function : Word
```

GetBkColorProc is the procedure prototype for the GetBkColor (674) method handler in TMod-Info (673). 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 (674) procedure prototype.

```
getpixelproc = function(X: SmallInt;Y: SmallInt) : Word
```

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 (674)

```
getscanlineproc = procedure(X1: SmallInt;X2: SmallInt;Y: SmallInt;
                            var data)
```

This routine is used for FloodFill (678) 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.

```
graphgetmemprc = 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 (674) 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.

```
LineStyleType = record  
  linestyle : Word;  
  pattern : Word;  
  thickness : Word;  
end
```

Record describing current line drawing mode

```
OutTextXYProc = procedure (x: SmallInt; y: SmallInt;  
                           const TextString: string)
```

This routine prototype is a hook for OutTextXY (675)

```
PaletteType = record  
  Size : LongInt;  
  Colors : Array[0..MaxColors] of RGBRec;  
end
```

Record describing palette.

```
patternlineproc = procedure(x1: SmallInt;x2: SmallInt;y: SmallInt)
```

Standard procedure prototype to draw a patterned line

```
PCharsetTransTable = ^TCharsetTransTable
```

Pointer to TCharsetTransTable (672) array.

```
PModeInfo = ^TModeInfo
```

Pointer to TModeInfo (673) record

```
PointType = record
  x : SmallInt;
  y : SmallInt;
end
```

Record describing a point in a 2 dimensional plane

```
putimageproc = procedure(X: SmallInt;Y: SmallInt;var Bitmap;
                          BitBlt: Word)
```

Standard PutImage (675) procedure prototype.

```
putpixelproc = procedure(X: SmallInt;Y: SmallInt;Color: Word)
```

Standard pixel drawing routine prototype

```
restorestateproc = procedure
```

Standard routine prototype to restore the graphical state at a closegraph call.

```
RGBRec = packed record
  Red : SmallInt;
  Green : SmallInt;
  Blue : SmallInt;
end
```

Record describing palette RGB color

```
savestateproc = procedure
```

Standard routine prototype to save the graphical state before a mode is set.

```
setactivepageproc = procedure(page: Word)
```

Standard SetActivePage (675) procedure prototype.

```
SetAllPaletteProc = procedure(const Palette: PaletteType)
```

This routine prototype is a hook for SetAllPalette (675)

```
SetBkColorProc = procedure(ColorNum: Word)
```

SetBkColorProc is the procedure prototype for the SetBkColor (675) method handler in TMod-Info (673). 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 (676)

```
setvisualpageproc = procedure(page: Word)
```

Standard SetVisualPage (676) procedure prototype.

```
TCharsetTransTable = Array[Char] of Char
```

Character transliteration table, with entries for 256 characters

```
TextSettingsType = record
    font : Word;
    direction : Word;
    charsize : Word;
    horiz : Word;
    vert : Word;
end
```

Record describing how texts are drawn.

```
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.

```

TResolutionRec = record
  x : LongInt;
  y : LongInt;
end

```

Record describing resolution

```

ViewPortType = record
  x1 : SmallInt;
  y1 : SmallInt;
  x2 : SmallInt;
  y2 : SmallInt;
  Clip : Boolean;
end

```

Record describing a viewport

```
vlineproc = procedure(x: SmallInt;y: SmallInt;y2: SmallInt)
```

Standard procedure prototype to draw a single vertical line

18.12.3 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

18.13 Procedures and functions

18.13.1 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 (673), Ellipse (678), GetArcCoords (679), PieSlice (688), Sector (689)

18.13.2 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 (676), Rectangle (688)

18.13.3 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 (676), Rectangle (688)

18.13.4 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` ([674](#)), `SetBkColor` ([675](#))

18.13.5 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` ([685](#))

18.13.6 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` ([685](#))

18.13.7 DrawPoly

Synopsis: Draw a polygone

Declaration: `procedure DrawPoly(NumPoints: Word; var polypoints)`

Visibility: `default`

Description: `DrawPoly` draws a polygone with `NumberOfPoints` corner points, using the current color and line-style. `PolyPoints` is an array of type `PointType` ([671](#)).

Errors: None.

See also: `Bar` ([676](#)), `Bar3D` ([676](#)), `Rectangle` ([688](#))

18.13.8 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 \(676\)](#), [Circle \(673\)](#), [FillEllipse \(678\)](#)

18.13.9 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 \(676\)](#), [Circle \(673\)](#), [GetArcCoords \(679\)](#), [PieSlice \(688\)](#), [Sector \(689\)](#)

18.13.10 FillPoly

Synopsis: Draw, close and fill a polygone

Declaration: `procedure FillPoly(NumPoints: Word;var PolyPoints)`

Visibility: default

Description: `FillPoly` draws a polygone 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 \(676\)](#), [Bar3D \(676\)](#), [Rectangle \(688\)](#)

18.13.11 FloodFill

Synopsis: Fill an area with a given color

Declaration: `procedure FloodFill(x: SmallInt;y: SmallInt;Border: Word)`

Visibility: default

Description: Fills the area containing the point (X, Y) , bounded by the color `BorderColor`.

Errors: None

See also: [SetColor \(690\)](#), [SetBkColor \(675\)](#)

18.13.12 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` ([676](#)), `Ellipse` ([678](#))

18.13.13 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` ([685](#)), `SetAspectRatio` ([689](#))

18.13.14 GetColor

Synopsis: Return current drawing color

Declaration: `function GetColor : Word`

Visibility: default

Description: `GetColor` returns the current drawing color (the palette entry).

Errors: None.

See also: `GetColor` ([679](#)), `SetBkColor` ([675](#))

18.13.15 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` ([679](#)), `GetBkColor` ([674](#))

18.13.16 GetDirectVideo

Synopsis: Determine whether direct video mode is active.

Declaration: `function GetDirectVideo : Boolean`

Visibility: default

Description: Determine whether direct video mode is active.

18.13.17 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` ([682](#)), `InitGraph` ([685](#))

18.13.18 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` ([690](#))

18.13.19 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` ([690](#))

18.13.20 GetGraphMode

Synopsis: Get current graphical modus

Declaration: `function GetGraphMode : SmallInt`

Visibility: default

Description: `GetGraphMode` returns the current graphical modus

Errors: None.

See also: `InitGraph` ([685](#))

18.13.21 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` ([691](#))

18.13.22 GetMaxColor

Synopsis: return maximum number of colors

Declaration: `function GetMaxColor : Word`

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` ([690](#)), `GetPaletteSize` ([683](#))

18.13.23 GetMaxMode

Synopsis: Return biggest mode for the current driver

Declaration: `function GetMaxMode : SmallInt`

Visibility: default

Description: `GetMaxMode` returns the highest modus for the current driver.

Errors: None.

See also: `InitGraph` ([685](#))

18.13.24 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` ([682](#))

18.13.25 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` ([682](#))

18.13.26 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` ([680](#)), `InitGraph` ([685](#))

18.13.27 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` ([685](#))

18.13.28 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` ([683](#)), `SetPalette` ([692](#))

18.13.29 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` ([683](#)), `SetPalette` ([692](#))

18.13.30 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` ([692](#)), `SetTextJustify` ([692](#))

18.13.31 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` ([693](#))

18.13.32 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` ([684](#))

18.13.33 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` ([684](#))

18.13.34 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` ([693](#)), `SetFillStyle` ([690](#)), `SetColor` ([690](#)), `SetBkColor` ([675](#)), `SetLineStyle` ([691](#))

18.13.35 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` ([1](#))

18.13.36 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` (686)
- `SetLineStyle` (691)
- `SetWriteMode` (694)
- `SetFillStyle` (690)
- `SetTextJustify` (692)
- `SetGraphMode` (691)
- `SetTextStyle` (692)

Errors: None.

See also: `GraphErrorMsg` (1)

18.13.37 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 ([647](#)), DetectGraph ([677](#)), CloseGraph ([677](#)), GraphResult ([1](#))

18.13.38 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` ([685](#)), `InstallUserFont` ([686](#))

18.13.39 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` ([685](#)), `InstallUserDriver` ([686](#))

18.13.40 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` ([675](#)), `LineTo` ([687](#))

18.13.41 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` ([686](#)), `Line` ([675](#))

18.13.42 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` ([687](#))

18.13.43 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` ([687](#))

18.13.44 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` ([675](#))

18.13.45 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 ([676](#)), Circle ([673](#)), Sector ([689](#))

18.13.46 queryadapterinfo

Synopsis: Function called to retrieve the current video adapter settings.

Declaration: `function queryadapterinfo : PModeInfo`

Visibility: default

18.13.47 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 ([676](#)), Bar3D ([676](#))

18.13.48 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 ([686](#)), RegisterBGIFont ([689](#))

18.13.49 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` ([686](#)), `RegisterBGIDriver` ([688](#))

18.13.50 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` ([685](#))

18.13.51 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` ([676](#)), `Circle` ([673](#)), `PieSlice` ([688](#))

18.13.52 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` ([685](#)), `GetAspectRatio` ([679](#))

18.13.53 SetColor

Synopsis: Set foreground drawing color

Declaration: `procedure SetColor (Color: Word)`

Visibility: default

Description: Sets the foreground color to `Color`.

Errors: None.

See also: `GetColor` (679), `SetBkColor` (675), `SetWriteMode` (694)

18.13.54 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.

18.13.55 SetFillPattern

Synopsis: Set drawing fill pattern

Declaration: `procedure SetFillPattern (Pattern: FillPatternType; Color: Word)`

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` (680), `SetFillStyle` (690), `SetWriteMode` (694)

18.13.56 SetFillStyle

Synopsis: Set drawing fill style

Declaration: `procedure SetFillStyle (Pattern: Word; Color: Word)`

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 \(690\)](#), [SetWriteMode \(694\)](#)

18.13.57 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 \(685\)](#)

18.13.58 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 pre-defined 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 pre-defined constants:

NormWidth Normal line width

ThickWidth Double line width

Errors: None.

See also: [GetLineSettings \(681\)](#), [SetWriteMode \(694\)](#)

18.13.59 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` ([675](#)), `SetRGBPalette` ([676](#))

18.13.60 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 pre-defined constants:

LeftTextText is set left of the pointer.

CenterTextText is set centered horizontally on the pointer.

RightTextText 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 pre-defined constants :

BottomTextText is placed under the pointer.

CenterTextText is placed centered vertically on the pointer.

TopTextText is placed above the pointer.

Errors: None.

See also: `OutText` ([687](#)), `OutTextXY` ([675](#))

18.13.61 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. pre-defined constants for `Font` are:

DefaultFontThe default font

TriplexFontA special font

SmallFontA smaller font

SansSerifFontA sans-serif font (like Arial)

GothicFontA gothic font

ScriptFontA script font

SimpleFontA simple font

TSCRFonTerminal screen font

LCOMFont?

EuroFont?

BoldFontA bold typeface font

Pre-defined constants for `Direction` are :

HorizDirWrite horizontal

VertDirWrite vertical

Errors: None.

See also: `GetTextSettings` ([683](#))

18.13.62 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` ([692](#))

18.13.63 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` ([683](#))

18.13.64 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 pre-defined constants:

CopyPutDraw as specified using current bitmask and color

XORPutDraw XOR-ing current bitmask and color

Errors: None.

See also: `SetColor` ([690](#)), `SetBkColor` ([675](#)), `SetLineStyle` ([691](#)), `SetFillStyle` ([690](#))

18.13.65 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` ([694](#))

18.13.66 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` ([694](#))

Chapter 19

Reference for unit 'heaptrc'

19.1 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.

19.2 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` (697) variable is set to `True`

disabled If this string occurs, then the `UseHeapTrace` (698) 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` (697) 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` (700))

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.

19.3 HeapTrc Usage

All that you need to do is to include `heaptrc` in the `uses` clause of your program. Make sure that it is the first unit in the clause, otherwise memory allocated in initialization code of units that precede the `heaptrc` unit will not be accounted for, causing an incorrect memory usage report.

If you use the `-gh` switch, the compiler will insert the unit by itself, so you don't have to include it in your `uses` clause.

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: `./heapex/heapex.pp`

Program `heapex`;

{ Program used to demonstrate the usage of heaptrc unit }

Uses `heaptrc`;

Var `P1` : ^Longint;
 `P2` : Pointer;
 `I` : longint;

begin

```
  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 (1 mod 2) = 0 Then FreeMem(P2,128);
    end;
    GetMem(P2,128);
    // This will provoke an error and a memory dump
    Freemem (P2,64);
end.

```

19.4 Constants, types and variables

19.4.1 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.

```
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` (699) 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` (698) or `printfaultyblock` (697) 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` (697) 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` (697) 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`.

```
tracesize = 8
```

`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 must be set at program startup, through the help of an environment variable.

19.4.2 Types

```
tdisplayextrainfoProc = procedure(var ptext: text;p: pointer)
```

The `TDisplayExtraInfoType` is a procedural type used in the `SetHeapExtraInfo` (699) call to display a memory location which was previously filled with `TFillExtraInfoProc` (698)

```
tFillExtraInfoProc = procedure(p: pointer)
```

The `TFillExtraInfoProc` is a procedural type used in the `SetHeapExtraInfo` (699) call to fill a memory location with extra data for displaying.

19.5 Procedures and functions

19.5.1 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.

19.5.2 DumpHeap

Synopsis: Dump memory usage report to stderr.

Declaration: `procedure DumpHeap`

Visibility: default

Description: `DumpHeap` dumps to standard output 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.

19.5.3 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` (699) is executed.

See also: `DumpHeap` (699), `SetHeapTraceOutput` (700)

Listing: `./heapex/setinfo.pp`

Program `heapex`;

{ Program used to demonstrate the usage of heaptrc unit }

Uses `heaptrc`;

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.

```

19.5.4 SetHeapTraceOutput

Synopsis: Specify filename for heap trace output.

Declaration: `procedure SetHeapTraceOutput(const name: string); Overload`
`procedure SetHeapTraceOutput(var ATextOutput: Text); Overload`

Visibility: default

Description: `SetHeapTraceOutput` sets the filename into which heap trace info will be written. By default information is written to standard output, this function allows you to redirect the information to a file with full filename name.

Errors: If the file cannot be written to, errors will occur when writing the trace.

See also: `SetHeapExtraInfo` ([699](#))

Chapter 20

Reference for unit 'ipc'

20.1 Used units

Table 20.1: Used units by unit 'ipc'

Name	Page
BaseUnix	100
System	1118
unixtype	1623

20.2 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.

20.3 Constants, types and variables

20.3.1 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_RMID = 0

Remove resource

IPC_SET = 1

set ipc_perm options

IPC_STAT = 2

get ipc_perm options

MSG_NOERROR = 1 shl 12

Internal Message control code. Do not use

SEM_A = 2 shl 6

Constant to describe alter permissions on Semaphores

SEM_R = 4 shl 6

Constant to describe read permissions on Semaphores

SEM_UNDO = 1 shl 12

Constant for use in semop ([716](#))

SHM_LOCK = 11

This constant is used in the shmctl ([717](#)) call.

SHM_R = IPC_R

This constant is used in the shmctl ([717](#)) call.

SHM_RDONLY = 1 shl 12

This constant is used in the shmctl ([717](#)) call.

SHM_RND = 2 shl 12

This constant is used in the shmctl ([717](#)) call.

SHM_UNLOCK = 12

This constant is used in the shmctl ([717](#)) call.

SHM_W = IPC_W

This constant is used in the shmctl ([717](#)) call.

20.3.2 Types

`key_t = TKey`

Alias for TKey (704) type

`msglen_t = culong`

Message length type

`msgqnum_t = culong`

Message queue number type

`PIPC_Perm = ^TIPC_Perm`

Pointer to TIPC_Perm (704) record.

`PMSG = ^TMSG`

Pointer to TMSG (704) record

`PMSGbuf = ^TMSGbuf`

Pointer to TMsgBuf (704) record

`PMSGinfo = ^TMSGinfo`

Pointer to TMSGinfo (705) record

`PMSQid_ds = ^TMSQid_ds`

Pointer to TMSQid_ds (705)

`PSEM = ^TSEM`

Pointer to TSEM

`PSEMbuf = ^TSEMbuf`

Pointer to TSembuf (705) record.

`PSEMid_ds = ^TSEMid_ds`

Pointer to TSEMid_ds (706) record.

`PSEMinfo = ^TSEMinfo`

Pointer to TSEMinfo (706) record.

`PSEMun = ^TSEMun`

Pointer to TSEMun (706) record

```
PShmid_DS = ^TShmid_ds
```

Pointer to TSHMid_ds (701) record.

```
PSHMinfo = ^TSHMinfo
```

```
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;
  seq : cushort;
end
```

TIPC_Perm is used in all IPC systems to specify the permissions. It should never be used directly.

```
TKey = cint
```

Type returned by the ftok (706) key generating function.

```
TMSG = record
end
```

Record used in the handling of message queues. Do not use directly.

```
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 accomodate your messages. The mtype field should always be present, and should always be filled.

```
TMSGinfo = record
  msgmax : cint;
  msgmni : cint;
  msgmnb : cint;
  msgtql : cint;
  msgssz : cint;
  msgseg : cint;
end
```

Internal message system record. Do not use directly.

```
TMSQid_ds = record
  msg_perm : TIPC_Perm;
  msg_first : PMSG;
  msg_last : PMSG;
  msg_cbytes : msglen_t;
  msg_qnum : msgqnum_t;
  msg_qbytes : msglen_t;
  msg_lspid : pid_t;
  msg_lrpid : pid_t;
  msg_stime : time_t;
  msg_pad1 : clong;
  msg_rtime : time_t;
  msg_pad2 : clong;
  msg_ctime : time_t;
  msg_pad3 : clong;
  msg_pad4 : Array[0..3] of clong;
end
```

This record should never be used directly, it is an internal kernel record. It's fields may change at any time.

```
TSEM = record
end
```

TSEM is a semaphore description record. It should be considered opaque, it's structure is highly OS and CPU dependent.

```
TSEMBuf = record
  sem_num : cushort;
  sem_op : cshort;
  sem_flg : cshort;
end
```

The TSEMBuf record is used in the semop (716) call, and is used to specify which operations you want to do.

```
TSEMid_ds = record
  sem_perm : TIPC_Perm;
  sem_base : PSEM;
  sem_nsems : cushort;
  sem_otime : time_t;
  sem_pad1 : cint;
  sem_ctime : time_t;
  sem_pad2 : cint;
  sem_pad3 : Array[0..3] of cint;
end
```

Structure returned by the `semctl` (711) call, contains all data of a semaphore

```
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.

```
TSEMun = record
end
```

Record used in `semctl` (711) call.

```
TSHMinfo = record
  shmmax : cint;
  shmmni : cint;
  shmmns : cint;
  shmseg : cint;
  shmall : cint;
end
```

Record used by the shared memory system, Do not use directly.

20.4 Procedures and functions

20.4.1 `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` (715), `shmget` (720) or `msgget` (709) 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` (707), `semctl` (711) or `shmctl` (717).

Errors: `ftok` returns -1 if the file in `Path` doesn't exist.

See also: `semget` (715), `shmget` (720), `msgget` (709)

20.4.2 msgctl

Synopsis: Perform various operations on a message queue

Declaration: `function msgctl(msqid: cint;cmd: cint;buf: PMSQid_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_STATIn this case, the `msgctl` call fills the `TMSQid_ds` structure with information about the message queue.

IPC_SETIn this case, the `msgctl` call sets the permissions of the queue as specified in the `ipc_perm` record inside `buf`.

IPC_RMIDIf 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 successful, `False` otherwise.

Errors: On error, `False` is returned, and `IPCError` is set accordingly.

See also: `msgget` (709), `msgsnd` (710), `msgrcv` (710)

Listing: `./ipccex/msgtool.pp`

program msgtool;

Uses ipc,baseunix;

Type

```
PMyMsgBuf = ^TMyMsgBuf;
TMyMsgBuf = record
  mtype : Longint;
  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,mtyp,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.

```

20.4.3 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_CREATThe queue is created if it doesn't already exist.

IPC_EXCLIf 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` (707).

Errors: On error, -1 is returned, and `IPCError` is set.

See also: `ftok` (706), `msgsnd` (710), `msgrcv` (710), `msgctl` (707)

20.4.4 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:

0No special meaning.

IPC_NOWAITIf no messages are available, then the call returns immediately, with the `ENOMSG` error.

MSG_NOERRORIf 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` (707).

Errors: In case of error, `False` is returned, and `IPCError` is set.

See also: `msgget` (709), `msgsnd` (710), `msgctl` (707)

20.4.5 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):

0No special meaning. The message will be written to the queue. If the queue is full, then the process is blocked.

IPC_NOWAITIf 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` (707).

Errors: In case of error, the call returns `False`, and `IPCError` is set.

See also: `msgget` (709), `msgrcv` (710), `msgctl` (707)

20.4.6 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_STATIn 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_SETIn 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_RMIDIf this is specified, the semaphore set is removed from the system.

GETALLIn 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 `\var{SizeOf(Word)* Number of semaphores in the set}`. This call will then fill the memory array with all the values of the semaphores.

GETNCNTThis 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` (716) 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.

SETALLIn 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 `\var{SizeOf(Word)* Number of semaphores in the set}`. This call will then set the values of the semaphores from the memory array.

SETVALThis 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 \(715\)](#), [semop \(716\)](#)

Listing: ./ipccex/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(etele) ' );
    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.

```

20.4.7 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_CREATThe queue is created if it doesn't already exist.

IPC_EXCLIf 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` (706), `semop` (716), `semctl` (711)

20.4.8 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` (705) structure

```
TSEMbuf = record
    sem_num : word;
    sem_op  : integer;
    sem_flg : integer;
```

should be filled as follows:

sem_numThe number of the semaphore in the set on which the operation must be performed.

sem_opThe 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_flgOptional 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` (715), `semctl` (711)

20.4.9 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` (717).

Errors: If an error occurs, -1 is returned, and `IPCError` is set.

See also: `shmget` (720), `shmdt` (719), `shmctl` (717)

20.4.10 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` (720), `shmat` (717), `shmdt` (719)

Listing: `./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 (shmid, 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 (shmid, 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'));
    shmid := shmget(key,segsz,IPC_CREAT or IPC_EXCL or 438);
    if shmid=-1 then
        begin
            writeln ( 'Shared memory exists. Opening as client');
            shmid := shmget(key,segsz,0);
            if shmid = -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;
    end.

```

20.4.11 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` (717).

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` (720), `shmat` (717), `shmctl` (717)

20.4.12 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.

Chapter 21

Reference for unit 'keyboard'

21.1 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` (735) function, which will return a driver-dependent key event. This key event can be translated to a interpretable event by the `TranslateKeyEvent` (744) 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` (743) function. The current keyboard driver can be retrieved using the `GetKeyboardDriver` (735) function. The last section of this chapter demonstrates how to make a keyboard driver.

21.2 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.

21.3 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` (735). Must wait for and return the next key event. It should NOT store keys.

PollKeyEvent Called by `PollKeyEvent` (740). It must return the next key event if there is one. Should not store keys.

GetShiftState Called by `PollShiftStateEvent` (741). 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: `./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;
  WriteIn(KeyLog,'Start logging keystrokes at: ',TimeStamp);
end;

Procedure StopKeyLogging;

begin
  WriteIn(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: ');
      WriteIn (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: ./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.
```

21.4 Keyboard scan codes

Special physical keys are encoded with the DOS scan codes for these keys in the second byte of the TKeyEvent (732) 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 21.1: 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 21.2: 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

21.5 Constants, types and variables

21.5.1 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', 'Midd`

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.

21.5.2 Types

```
PTreeElement = ^TTreeElement
```

Pointer to TTreeElement (733) record

```
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 (743) function.

The various fields correspond to the different functions of the keyboard unit interface. For more information about this record see kbddriver (722)

```
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 (732) type.

```
TKeyRecord = packed record
  Flags : Byte;
  ShiftState : Byte;
  KeyCode : Word;
end
```

The structure of a TKeyRecord structure is explained in the following table:

Table 21.3: Structure of TKeyRecord

Field	Meaning
KeyCode	Depending on <code>flags</code> either the physical representation of a key (under DOS scancode, ascii code pair), or the t
ShiftState	Shift-state when this key was pressed (or shortly after)
Flags	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).

```
Tprocedure = procedure
```

Procedure prototype

```
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` (733) on unix platforms. There should be no need to handle records of this type.

21.6 Procedures and functions

21.6.1 AddSequence

Declaration: `procedure AddSequence(const St: string; AChar: Byte; AScan: Byte)`

Visibility: default

21.6.2 AddSpecialSequence

Synopsis: Add a handler for a special key sequence

Declaration: `function AddSpecialSequence(const St: string; Proc: Tprocedure) : PTreeElement`

Visibility: default

Description: `AddSpecialSequence` adds a sequence of key combinations to the keyboard handler. When the key combination specified in `st` is encountered, then `Proc` will be executed. The function returns the element in the special key sequence handling tree which handles `st`.

See also: `AddSequence` ([733](#))

21.6.3 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` ([739](#)). 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` ([739](#))

21.6.4 FindSequence

Declaration: `function FindSequence(const St: string; var AChar: Byte; var Ascan: Byte) : Boolean`

Visibility: `default`

21.6.5 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` ([744](#)), `KeyEventToString` ([740](#))

Listing: `./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.

```

21.6.6 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 `kbddriver` ([722](#)).

Errors: None.

See also: `SetKeyboardDriver` ([743](#))

21.6.7 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` ([740](#)).

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` ([744](#)) or `TranslateKeyEventUnicode` ([744](#)). 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 \(742\)](#), [PollKeyEvent \(740\)](#), [TranslateKeyEvent \(744\)](#), [TranslateKeyEventUnicode \(744\)](#)

Listing: ./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.
```

21.6.8 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 \(735\)](#)

Errors: None.

See also: [GetKeyEventUnicode \(738\)](#), [GetKeyEventShiftState \(738\)](#), [GetKeyEventFlags \(737\)](#), [GetKeyEventCode \(736\)](#), [GetKeyEvent \(735\)](#)

21.6.9 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` (738), `GetKeyEventShiftState` (738), `GetKeyEventFlags` (737), `GetKeyEventChar` (736), `GetKeyEvent` (735)

Listing: `./kbdex/ex2.pp`

Program `Example2`;

{ Program to demonstrate the GetKeyEventCode function. }

Uses `keyboard`;

Var

`K : TKeyEvent`;

begin

`InitKeyBoard`;

WriteIn ('Press function keys , or press "q" to end.');

Repeat

`K:=GetKeyEvent`;

`K:=TranslateKeyEvent(K)`;

If (`GetKeyEventFlags(K)<>KbfnKey`) **then**

WriteIn ('Not a function key')

else

begin

Write ('Got key (', `GetKeyEventCode(K)`);

WriteIn (') : ', `KeyEventToString(K)`);

end;

Until (`GetKeyEventChar(K)= 'q'`);

`DoneKeyboard`;

end.

21.6.10 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` (735)

Errors: None.

See also: `GetKeyEventUnicode` (738), `GetKeyEventShiftState` (738), `GetKeyEventCode` (736), `GetKeyEventChar` (736), `GetKeyEvent` (735)

21.6.11 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` (738), `GetKeyEventFlags` (737), `GetKeyEventCode` (736), `GetKeyEventChar` (736), `GetKeyEvent` (735)

Listing: `./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.

21.6.12 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` (738), `GetKeyEventFlags` (737), `GetKeyEventCode` (736), `GetKeyEventChar` (736), `GetKeyEvent` (735)

21.6.13 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` (734) function should also be called before exiting the program or changing the keyboard driver with `SetKeyboardDriver` (743).

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` (734), `SetKeyboardDriver` (743)

21.6.14 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` (735)

Listing: `./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 .

```

21.6.15 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` ([734](#)), `ShiftStateToString` ([744](#))

21.6.16 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` ([740](#)) and checks for a valid result.

Errors: None.

See also: `PollKeyEvent` ([740](#)), `GetKeyEvent` ([735](#))

21.6.17 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` ([735](#)) function.

Errors: None.

See also: [PutKeyEvent \(742\)](#), [GetKeyEvent \(735\)](#)

Listing: ./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.

```

21.6.18 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 \(740\)](#), [GetKeyEvent \(735\)](#)

Listing: ./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.

```

21.6.19 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` ([740](#)), `GetKeyEvent` ([735](#))

Listing: `./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.
```

21.6.20 RawReadKey

Declaration: function RawReadKey : Char

Visibility: default

21.6.21 RawReadString

Declaration: function RawReadString : string

Visibility: default

21.6.22 RestoreStartMode

Declaration: procedure RestoreStartMode

Visibility: default

21.6.23 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 ([734](#)).

The function returns True if the driver was set, False if not.

For more information on setting the keyboard driver, see kbddriver ([722](#)).

Errors: None.

See also: GetKeyboardDriver ([735](#)), DoneKeyboard ([734](#))

21.6.24 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` (741).

Errors: None.

See also: `FunctionKeyName` (734), `KeyEventToString` (740)

21.6.25 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` (735)

Errors: None.

See also: `TranslateKeyEventUnicode` (744)

21.6.26 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` (744)

Chapter 22

Reference for unit 'lineinfo'

22.1 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` ([773](#)) unit must be used.

22.2 Procedures and functions

22.2.1 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 succesful 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 23

Reference for unit 'Linux'

23.1 Used units

Table 23.1: Used units by unit 'Linux'

Name	Page
BaseUnix	100
System	1118
unixtype	1623

23.2 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` ([1589](#)), `baseunix` ([100](#)) and `unixutil` ([1638](#)) units.

The X86-specific parts have been moved to the X86 ([1668](#)) unit.

23.3 Constants, types and variables

23.3.1 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 (763) option: open files shared between processes

`CLONE_FS = $00000200`

Clone (763) option: fs info shared between processes

`CLONE_NEWNS = $00020000`

Clone options: Start child in new (filesystem) 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 (763) 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 (763) 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 (763) option: VM shared between processes

CSIGNAL = \$000000ff

Clone (763) 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 colour 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

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

PIO_CMAP = \$4B71

IOCTL: Set colour 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

```
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.

23.3.2 Types

```
clockid_t = cint
```

Clock id type

```
EPoll_Data = record
end
```

Data structure used in EPOLL IOCTL call.

```
EPoll_Event = record
  Events : cuint32;
  Data : TEPoll_Data;
end
```

Structure used in `epoll_ctl` ([766](#)) call.

```
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.

PEPoll_Data = ^EPoll_Data

Pointer to EPoll_Data (760) record

PEpoll_Event = ^EPoll_Event

Pointer to EPoll_Event (760) type

Pinotify_event = ^inotify_event

Pointer to inotify_event (760) structure.

PSysInfo = ^TSysInfo

Pointer to TSysInfo (762) record.

Puser_cap_data = ^user_cap_data

Pointer to user_cap_data (762) record

Puser_cap_header = ^user_cap_header

Pointer to user_cap_header (762) record

TCloneFunc = function(args: pointer) : LongInt

Clone function prototype.

TEPoll_Data = EPoll_Data

Alias for EPoll_Data (760) type

TEPoll_Event = EPoll_Event

Alias for EPoll_Event (760) type

```
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;
  _f : Array[0..19-2*sizeof(clong)-sizeof(cint)] of cchar;
end
```

Record with system information, used by the SysInfo ([771](#)) call.

```
user_cap_data = record
  effective : cuint32;
  permitted : cuint32;
  inheritable : cuint32;
end
```

`user_cap_data` describes the set of capabilities for the indicated thread.

```
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` ([762](#)) and `capset` ([762](#))

23.4 Procedures and functions

23.4.1 `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 `fperrno` is set to the error.

See also: `capset` ([762](#))

23.4.2 `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 `fperrno` is set to the error.

See also: `capget` ([762](#))

23.4.3 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` (763), `clock_settime` (763)

23.4.4 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` (763), `clock_settime` (763)

23.4.5 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` (763), `clock_gettime` (763)

23.4.6 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` (151) 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_VMParent and child share the same memory space, including memory (un)mapped with subsequent `mmap` calls.

CLONE_FSParent and child have the same view of the filesystem; the `chroot`, `chdir` and `umask` calls affect both processes.

CLONE_FILESthe file descriptor table of parent and child is shared.

CLONE_SIGHANDthe parent and child share the same table of signal handlers. The signal masks are different, though.

CLONE_PIDParent 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_eagainToo many processes are running.

sys_enomemNot enough memory to create child process.

See also: `#rtl.baseunix.FpFork` (151)

Listing: `./linuxex/ex71.pp`

```

program TestC{clone};

{ $ifdef Linux }
// close is very Linux specific. 1.9.x threading is done via pthreads.

uses
    Linux , Errors , crt ;

const
    Ready : Boolean = false ;
    aChar : Char    = 'a' ;

function CloneProc( Arg : Pointer ) : LongInt ; Cdecl ;
begin
    WriteLn('Hello from the clone ', PChar(Arg));
    repeat
        Write(aChar);
        Select(0,0,0,0,600);
    until Ready;
    WriteLn( 'Clone finished.' );
    CloneProc := 1;

```

```

end;

var
  PID : LongInt;

procedure MainProc;
begin
  WriteLn('cloned process PID: ', PID );
  WriteLn('Press <ESC> to kill ... ');
  repeat
    Write(' ');
    Select(0,0,0,0,300);
    if KeyPressed then
      case ReadKey of
        #27: Ready := true;
        'a': aChar := 'A';
        'A': aChar := 'a';
        'b': aChar := 'b';
        'B': aChar := 'B';
      end;
    until Ready;
    WriteLn('Ready. ');
  end;

  const
    StackSize = 16384;
    theFlags = CLONE_VM+CLONE_FS+CLONE_FILES+CLONE_SIGHAND;
    aMsg      : PChar = 'Oops !';

  var
    theStack : Pointer;
    ExitStat : LongInt;

  begin
    GetMem(theStack, StackSize);
    PID := Clone(@CloneProc,
                 Pointer( LongInt(theStack)+StackSize ),
                 theFlags,
                 aMsg);
    if PID < 0 then
      WriteLn('Error : ', LinuxError, ' when cloning.')
    else
      begin
        MainProc;
        case WaitPID(0, @ExitStat, Wait_Untraced or wait_clone) of
          -1: WriteLn('error: ', LinuxError, '; ', StrError(LinuxError));
          0: WriteLn('error: ', LinuxError, '; ', StrError(LinuxError));
        else
          WriteLn('Clone exited with: ', ExitStat shr 8);
        end;
      end;
      FreeMem( theStack, StackSize );
    {$else}
  begin
    {$endif}
  end.

```

23.4.7 `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` (766) or `epoll_wait` (767) calls, and should be closed using the `fpClose` (145) call.

Errors: On error, -1 is returned, and `errno` (154) is set.

See also: `epoll_ctl` (766), `epoll_wait` (767), `fpClose` (145)

23.4.8 `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_ADDAdd filedescriptor to list of events

EPOLL_CTL_MODModify event for filedescriptor

EPOLL_CTL_DELDelete event for filedescriptor

The `events` field in `event_data` is a bitmask of one or more of the following values:

EPOLLINThe file is ready for read operations

EPOLLOUTThe file is ready for write operations.

EPOLLPRIUrgent data is available for read operations.

EPOLLERRAn error condition is signaled on the file descriptor.

EPOLLHUPA Hang up happened on the file descriptor.

EPOLLETSet the Edge Triggered behaviour for the file descriptor.

EPOLLONESHOTSet 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` (766), `epoll_wait` (767), `fpClose` (145)

23.4.9 `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` (766), `epoll_ctl` (766), `fcntl` (145)

23.4.10 `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 `fsync` 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: `fcntl`.`fsync` (1589)

23.4.11 `futex`

Synopsis: Perform a `futex` operation

Declaration: `function futex(uaddr: pcint; op: cint; val: cint; timeout: timespec; addr2: pcint; val3: cint) : cint`
`function futex(var uaddr; op: cint; val: cint; timeout: timespec; 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: timespec) : cint`
`function futex(var uaddr; op: cint; val: cint; timeout: timespec) : 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_WAIT `Futex` option: Wait on `futex` till wake call arrives.

FUTEX_WAKE `Futex` 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` ([154](#)) to get the error code.

23.4.12 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);
```

23.4.13 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` (760) structure's `wd`.

Errors: On Error, -1 is returned. `fpgeterrno` can be used to get more detailed error information.

See also: `inotify_init` (769), `inotify_init1` (769), `inotify_rm_watch` (770), `inotify_event` (760)

23.4.14 `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` (769), `inotify_add_watch` (768), `inotify_rm_watch` (770)

23.4.15 `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_CLOEXEC`inotify` close on exec flag.

Errors: On Error, -1 is returned. `fpgeterrno` can be used to get more detailed error information.

See also: `inotify_init` (769), `inotify_add_watch` (768), `inotify_rm_watch` (770)

23.4.16 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` (769), `inotify_init1` (769), `inotify_add_watch` (768), `inotify_event` (760)

23.4.17 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.

23.4.18 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: `fdatasync` (767)

23.4.19 Sysinfo

Synopsis: Return kernel system information

Declaration: `function Sysinfo(Info: PSysInfo) : cint`

Visibility: default

Description: `SysInfo` returns system information in `Info`. 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: None.

See also: `#rtl.baseunix.fpUname` ([190](#))

Listing: `./linuxex/ex64.pp`

program Example64;

*{ 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) **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.
```

Chapter 24

Reference for unit 'Infodwrf'

24.1 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` ([745](#)) unit must be used.

24.2 Procedures and functions

24.2.1 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 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 25

Reference for unit 'math'

25.1 Used units

Table 25.1: Used units by unit 'math'

Name	Page
System	1118
sysutils	1360

25.2 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.

25.3 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 (791), NumberOfPeriods (807), Payment (807), PresentValue (810) and InterestRate (794) solve this equation for one of the variables, when the other variables are known.

See also: FutureValue (791), NumberOfPeriods (807), Payment (807), PresentValue (810), TPaymentTime (780)

25.4 Geometrical functions

Table 25.2:

Name	Description
hypot (793)	Hypotenuse of triangle
norm (806)	Euclidian norm

25.5 Statistical functions

Table 25.3:

Name	Description
mean (801)	Mean of values
meanandstddev (802)	Mean and standard deviation of values
momentskewkurtosis (805)	Moments, skew and kurtosis
popnstddev (808)	Population standarddeviation
popnvariance (808)	Population variance
randg (812)	Gaussian distributed random value
stddev (817)	Standard deviation
sum (817)	Sum of values
sumofsquares (818)	Sum of squared values
sumsandsquares (819)	Sum of values and squared values
totalvariance (821)	Total variance of values
variance (822)	variance of values

25.6 Number converting

Table 25.4:

Name	Description
<code>ceil</code> (784)	Round to infinity
<code>floor</code> (790)	Round to minus infinity
<code>frexp</code> (790)	Return mantissa and exponent

25.7 Exponential and logarithmic functions

Table 25.5:

Name	Description
<code>intpower</code> (794)	Raise float to integer power
<code>ldexp</code> (796)	Calculate $2^x \times \text{float}$
<code>lnxp1</code> (796)	calculate $\log(x+1)$
<code>log10</code> (797)	calculate 10-base log
<code>log2</code> (797)	calculate 2-base log
<code>logn</code> (798)	calculate N-base log
<code>power</code> (809)	raise float to arbitrary power

25.8 Hyperbolic functions

Table 25.6:

Name	Description
<code>arcosh</code> (781)	calculate reverse hyperbolic cosine
<code>arsinh</code> (783)	calculate reverse hyperbolic sine
<code>artanh</code> (784)	calculate reverse hyperbolic tangent
<code>cosh</code> (786)	calculate hyperbolic cosine
<code>sinh</code> (816)	calculate hyperbolic sine
<code>tanh</code> (821)	calculate hyperbolic tangent

25.9 Trigonometric functions

Table 25.7:

Name	Description
<code>arccos</code> (780)	calculate reverse cosine
<code>arcsin</code> (782)	calculate reverse sine
<code>arctan2</code> (782)	calculate reverse tangent
<code>cotan</code> (787)	calculate cotangent
<code>sincos</code> (815)	calculate sine and cosine
<code>tan</code> (820)	calculate tangent

25.10 Angle unit conversion

Routines to convert angles between different angle units.

Table 25.8:

Name	Description
<code>cycleto rad</code> (787)	convert cycles to radians
<code>degtograd</code> (788)	convert degrees to grads
<code>degtorad</code> (789)	convert degrees to radians
<code>gradtodeg</code> (792)	convert grads to degrees
<code>gradtora d</code> (792)	convert grads to radians
<code>radto cycle</code> (810)	convert radians to cycles
<code>radto deg</code> (811)	convert radians to degrees
<code>radto grad</code> (811)	convert radians to grads

25.11 Min/max determination

Functions to determine the minimum or maximum of numbers:

Table 25.9:

Name	Description
<code>max</code> (799)	Maximum of 2 values
<code>maxIntValue</code> (799)	Maximum of an array of integer values
<code>maxvalue</code> (800)	Maximum of an array of values
<code>min</code> (803)	Minimum of 2 values
<code>minIntValue</code> (803)	Minimum of an array of integer values
<code>minvalue</code> (804)	Minimum of an array of values

25.12 Constants, types and variables

25.12.1 Constants

`EqualsValue` = 0

Values are the same

`GreaterThanValue` = (TValueRelationship)

First values is greater than second value

`Infinity` = 1.0 / 0.0

Value is infinity

`LessThanValue` = (TValueRelationship)

First value is less than second value

`MaxComp` = 9.223372036854775807e + 18

`MaxDouble` = 1.7e + 308

`MaxExtended` = 1.1e + 4932

Maximum value of extended type

`MaxFloat` = 0

Maximum value of float type

`MaxSingle` = 3.4e + 38

`MinComp` = -9.223372036854775807e + 18

`MinDouble` = 5.0e - 324

`MinExtended` = 3.4e - 4932

Minimum value (closest to zero) of extended type

`MinFloat` = 0

Minimum value (closest to zero) of float type

`MinSingle` = 1.5e - 45

`NaN = 0.0 / 0.0`

Value is Not a Number

`NegativeValue = (TValueSign)`

Value is negative

`NegInfinity = (-1.0) / 0.0`

Value is negative (minus) infinity

`PositiveValue = (TValueSign)`

Value is positive

`ZeroValue = 0`

Value is zero

25.12.2 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` (779) is used in functions that accept an array of values of arbitrary length.

`PFloat = ^Float`

Pointer to `Float` (779) 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 25.10: 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` (813) function.

`TValueRelationship = -1..1`

Type to describe relational order between values

`TValueSign = -1..1`

Type indicating sign of a valuea

25.13 Procedures and functions

25.13.1 arccos

Synopsis: Return inverse cosine

Declaration: `function arccos(x: Float) : Float`

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` (782), `arcosh` (781), `arsinh` (783), `artanh` (784)

Listing: `./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.

```

25.13.2 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` (781), provided for Delphi compatibility.

See also: `arcosh` (781)

25.13.3 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` (786), `sinh` (816), `arcsin` (782), `arsinh` (783), `artanh` (784), `tanh` (821)

Listing: `./mathex/ex3.pp`

Program Example3;

{ Program to demonstrate the arcosh function. }

Uses math;

```

begin
  Writeln ( arcosh (1));
  Writeln ( arcosh (2));
end.

```

25.13.4 arcsin

Synopsis: Return inverse sine

Declaration: `function arcsin(x: Float) : Float`

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` (780), `arcosh` (781), `arsinh` (783), `artanh` (784)

Listing: `./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.

25.13.5 arcsinh

Synopsis: Return inverse hyperbolic sine

Declaration: `function arcsinh(x: Float) : Float`

Visibility: default

Description: `arcsinh` returns the inverse hyperbolic sine of its argument `x`.

This function is an alias for `arsinh` (783), provided for Delphi compatibility.

See also: `arsinh` (783)

25.13.6 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` (780), `arcosh` (781), `arsinh` (783), `artanh` (784)

Listing: `./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.

25.13.7 arctanh

Synopsis: Return inverse hyperbolic tangent

Declaration: `function arctanh(x: Float) : Float`

Visibility: default

Description: `arcsinh` returns the inverse hyperbolic tangent of its argument x .

This function is an alias for `artanh` (784), provided for Delphi compatibility.

See also: `artanh` (784)

25.13.8 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 `arcsinh` variant of this function is supplied for Delphi compatibility.

Errors: None.

See also: `arcosh` (781), `arccos` (780), `arcsin` (782), `artanh` (784)

Listing: `./mathex/ex4.pp`

```

Program Example4;

{ Program to demonstrate the arsinh function. }

Uses math;

begin
    Writeln(arsinh(0));
    Writeln(arsinh(1));
end.

```

25.13.9 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` (781), `arccos` (780), `arcsin` (782), `artanh` (784)

Listing: `./mathex/ex5.pp`

```

Program Example5;

{ Program to demonstrate the artanh function. }

Uses math;

begin
    Writeln(artanh(0));
    Writeln(artanh(0.5));
end.

```

25.13.10 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` (790)

Listing: `./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.
```

25.13.11 ClearExceptions

Synopsis: Clear Floating Point Unit exceptions

Declaration: `procedure ClearExceptions(RaisePending: Boolean)`

Visibility: default

Description: Clear Floating Point Unit exceptions

25.13.12 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 (780)

25.13.13 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` ([814](#))

25.13.14 cosh

Synopsis: Return hyperbolic cosine

Declaration: `function cosh (x: Float) : Float`

Visibility: default

Description: `Cosh` returns the hyperbolic cosine of its argument `{x}`.

Errors: None.

See also: `arcosh` ([781](#)), `sinh` ([816](#)), `arsinh` ([783](#))

Listing: `./mathex/ex8.pp`

Program `Example8`;

{ Program to demonstrate the cosh function. }

Uses `math`;

begin

WriteLn (`Cosh (0)`);

WriteLn (`Cosh (1)`);

end.

25.13.15 cot

Synopsis: Alias for `Cotan`

Declaration: `function cot (x: Float) : Float`

Visibility: default

Description: `cot` is an alias for the `cotan` ([787](#)) function.

See also: `cotan` ([787](#))

25.13.16 cotan

Synopsis: Return cotangent

Declaration: `function cotan(x: Float) : Float`

Visibility: default

Description: `Cotan` returns the cotangent of it's 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` (821)

Listing: `./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.

25.13.17 csc

Synopsis: Alias for cosecant

Declaration: `function csc(x: Float) : Float`

Visibility: default

Description: `csc` is an alias for the cosecant (786) function.

See also: `cosecant` (786)

25.13.18 cycletorad

Synopsis: Convert cycle angle to radians angle

Declaration: `function cycletorad(cycle: Float) : Float`

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` (788), `degtorad` (789), `radtodeg` (811), `radtograd` (811), `radto cycle` (810)

Listing: `./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.

25.13.19 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.

25.13.20 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 ([787](#)), degtorad ([789](#)), radto deg ([811](#)), radto grad ([811](#)), radto cycle ([810](#))

Listing: ./mathex/ex11.pp

Program Example11;

{ Program to demonstrate the degtograd function. }

Uses math;

begin

writeln(degtograd(90));

writeln(degtograd(180));

writeln(degtograd(270))

end.

25.13.21 degtorad

Synopsis: Convert degree angle to radians angle.

Declaration: `function degtorad(deg: Float) : Float`

Visibility: default

Description: Degtorad converts it's argument deg (an angle in degrees) to radians. (pi radians is 180 degrees)

Errors: None.

See also: [cycletorad \(787\)](#), [degtograd \(788\)](#), [radtodeg \(811\)](#), [radtograd \(811\)](#), [radto cycle \(810\)](#)

Listing: `./mathex/ex12.pp`

Program Example12;

{ Program to demonstrate the degtorad function. }

Uses math;

```
begin
  writeln (degtorad (45));
  writeln (degtorad (90));
  writeln (degtorad (180));
  writeln (degtorad (270));
  writeln (degtorad (360));
end.
```

25.13.22 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

25.13.23 EnsureRange

Synopsis: Change value to it falls in 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` ([794](#))

25.13.24 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` ([784](#))

Listing: `./mathex/ex13.pp`

Program `Example13;`

{ Program to demonstrate the floor function. }

Uses `math;`

begin

`WriteLn (Ceil (-3.7)); // should be -4`

`WriteLn (Ceil (3.7)); // should be 3`

`WriteLn (Ceil (-4.0)); // should be -4`

end.

25.13.25 Frexp

Synopsis: Return mantissa and exponent.

Declaration: `procedure Frexp(X: Float; var Mantissa: Float; var Exponent: Integer)`

Visibility: default

Description: `Frexp` returns the mantissa and exponent of it's argument `x` in mantissa and exponent.

Errors: None

Listing: `./mathex/ex14.pp`

Program `Example14;`

{ Program to demonstrate the frexp function. }

Uses `math;`

Procedure `dofrexp(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
//    dofrep(1.00);
    dofrep(1.02e-1);
    dofrep(1.03e-2);
    dofrep(1.02e1);
    dofrep(1.03e2);
end.

```

25.13.26 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 \(774\)](#)) 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 \(794\)](#), [NumberOfPeriods \(807\)](#), [Payment \(807\)](#), [PresentValue \(810\)](#), [TPaymentTime \(780\)](#), [CashFlowFunctions \(774\)](#)

25.13.27 GetExceptionMask

Synopsis: Get the Floating Point Unit exception mask.

Declaration: `function GetExceptionMask : TFPUExceptionMask`

Visibility: default

Description: Get the Floating Point Unit exception mask.

25.13.28 GetPrecisionMode

Synopsis: Return the Floating Point Unit precision mode.

Declaration: `function GetPrecisionMode : TFPUPrecisionMode`

Visibility: default

Description: Return the Floating Point Unit precision mode.

25.13.29 GetRoundMode

Synopsis: Return the Floating Point Unit rounding mode.

Declaration: `function GetRoundMode : TFPURoundingMode`

Visibility: default

Description: Return the Floating Point Unit rounding mode.

25.13.30 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` ([787](#)), `degtograd` ([788](#)), `radtodeg` ([811](#)), `radtograd` ([811](#)), `radtcycle` ([810](#)), `gradtorad` ([792](#))

Listing: `./mathex/ex15.pp`

Program `Example15;`

`{ Program to demonstrate the gradtodeg function. }`

Uses `math;`

begin

`writeln(gradtodeg(100));`

`writeln(gradtodeg(200));`

`writeln(gradtodeg(300));`

end.

25.13.31 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: `cycletorad` ([787](#)), `degtograd` ([788](#)), `radtodeg` ([811](#)), `radtograd` ([811](#)), `radtcycle` ([810](#)), `gradtodeg` ([792](#))

Listing: ./mathex/ex16.pp

Program Example16;

{ Program to demonstrate the gradtorad function. }

Uses math;

begin
 writeln(gradtorad(100));
 writeln(gradtorad(200));
 writeln(gradtorad(300));
end.

25.13.32 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: ./mathex/ex17.pp

Program Example17;

{ Program to demonstrate the hypot function. }

Uses math;

begin
 WriteLn(hypot(3,4)); *// should be 5*
end.

25.13.33 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.

25.13.34 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` ([789](#))

25.13.35 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` ([774](#))). 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` ([791](#)), `NumberOfPeriods` ([807](#)), `Payment` ([807](#)), `PresentValue` ([810](#)), `TPaymentTime` ([780](#)), `CashFlowFunctions` ([774](#))

25.13.36 intpower

Synopsis: Return integer power.

Declaration: `function intpower(base: Float;const exponent: Integer) : 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` ([809](#))

Listing: `./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.

25.13.37 IsInfinite

Synopsis: Check whether value is infinite

Declaration: function IsInfinite(const d: Double) : Boolean

 Visibility: default

Description: IsInfinite returns True if the double d contains the infinite value.

See also: IsZero ([795](#)), IsInfinite ([795](#))

25.13.38 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 ([795](#)), IsInfinite ([795](#))

25.13.39 IsZero

Synopsis: Check whether value is zero

See also: [ldexp \(796\)](#), [log10 \(797\)](#), [log2 \(797\)](#), [logn \(798\)](#)

Listing: ./mathex/ex20.pp

Program Example20;

{ Program to demonstrate the Inxp1 function. }

Uses math;

begin

writeln (Inxp1 (0));
 writeln (Inxp1 (0.5));
 writeln (Inxp1 (1));

end.

25.13.42 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 \(796\)](#), [lnxp1 \(796\)](#), [log2 \(797\)](#), [logn \(798\)](#)

Listing: ./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.

25.13.43 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` (796), `lnxp1` (796), `log10` (797), `logn` (798)

Listing: ./mathex/ex22.pp

Program Example22;

{ Program to demonstrate the log2 function. }

Uses math;

begin

WriteIn(Log2(2):8:4);

WriteIn(Log2(4):8:4);

WriteIn(Log2(8):8:4);

WriteIn(Log2(1):8:4);

WriteIn(Log2(0.5):8:4);

WriteIn(Log2(0.25):8:4);

WriteIn(Log2(0.125):8:4);

end.

25.13.44 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` (796), `lnxp1` (796), `log10` (797), `log2` (797)

Listing: ./mathex/ex23.pp

Program Example23;

{ Program to demonstrate the logn function. }

Uses math;

begin

WriteIn(Logn(3,4):8:4);

WriteIn(Logn(2,4):8:4);

WriteIn(Logn(6,9):8:4);

WriteIn(Logn(`exp`(1),`exp`(1)):8:4);

WriteIn(Logn(0.5,1):8:4);

WriteIn(Logn(0.25,3):8:4);

WriteIn(Logn(0.125,5):8:4);

end.

25.13.45 Max

Synopsis: Return largest of 2 values

Declaration: `function Max(a: Integer;b: Integer) : Integer; Overload`
`function Max(a: Int64;b: Int64) : Int64; 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 maximum of `Int1` and `Int2`.

Errors: None.

See also: `min` (803), `maxIntValue` (799), `maxvalue` (800)

Listing: `./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.

25.13.46 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` (800) function instead.

Errors: None.

See also: `maxvalue` (800), `minvalue` (804), `minIntValue` (803)

Listing: `./mathex/ex25.pp`

Program `Example25;`

{ Program to demonstrate the MaxIntValue function. }

{ Make sure 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.

25.13.47 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 (799), minvalue (804), minIntValue (803)

Listing: ./mathex/ex26.pp

program Example26;

{ Program to demonstrate the MaxValue 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);

WriteIn( 'Max Float      : ', MaxValue(f_array):8:4);
WriteIn( 'Max Float  (b) : ', MaxValue(Pf_array, 100):8:4);
WriteIn( 'Max Integer   : ', MaxValue(i_array):8);
WriteIn( 'Max Integer (b) : ', MaxValue(Pi_array, 100):8);
end.

```

25.13.48 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

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 ([802](#)), momentskewkurtosis ([805](#)), sum ([817](#))

Listing: ./mathex/ex27.pp

Program Example27;

{ Program to demonstrate the Mean function. }

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;
WriteIn('Max : ', MaxValue(ExArray):8:4);
WriteIn('Min : ', MinValue(ExArray):8:4);
WriteIn('Mean : ', Mean(ExArray):8:4);

```

    WriteLn ( 'Mean (b) : ', Mean(@ExArray[1], 100):8:4);
end.

```

25.13.49 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` (801), `sum` (817), `sumofsquares` (818), `momentskewkurtosis` (805)

Listing: `./mathex/ex28.pp`

Program `Example28`;

```
{ Program to demonstrate the Meanandstddev function. }
```

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.

25.13.50 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: 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 value of `Int1` and `Int2`;

Errors: None.

See also: `max` ([799](#))

Listing: `./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.

25.13.51 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` ([804](#)), `maxIntValue` ([799](#)), `maxvalue` ([800](#))

Listing: `./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.

25.13.52 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 (799), maxvalue (800), minIntValue (803)

Listing: ./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(l)-random(100);

WriteIn ('Min Float      : ', MinValue(f_array):8:4);
WriteIn ('Min Float   (b) : ', MinValue(Pf_array,100):8:4);
WriteIn ('Min Integer   : ', MinValue(i_array):8);
WriteIn ('Min Integer (b) : ', MinValue(Pi_array,100):8);
end.

```

25.13.53 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 values in data and returns them in m1,m2,m3 and m4, as well as the skew and kurtosis.

Errors: None.

See also: mean ([801](#)), meanandstddev ([802](#))

Listing: ./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.

```

25.13.54 norm

Synopsis: Return Euclidian 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 Euclidian 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` ([818](#))

Listing: ./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.

```

25.13.55 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` (774)). 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` (791), `InterestRate` (794), `Payment` (807), `PresentValue` (810), `TPaymentTime` (780), `CashFlowFunctions` (774)

25.13.56 operator **(Float, Float): Float

Declaration: `operator operator ** (Float, Float) : Float (bas: Float; expo: Float)
: Float`

Visibility: default

25.13.57 operator **(Int64, Int64): Int64

Declaration: `operator operator ** (Int64, Int64) : Int64 (bas: Int64; expo: Int64)
: Int64`

Visibility: default

25.13.58 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` (774)). 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 (791), InterestRate (794), NumberOfPeriods (807), PresentValue (810), TPaymentTime (780), CashFlowFunctions (774)

25.13.59 popnstddev

Synopsis: Return population variance

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 (808), mean (801), meanandstddev (802), stddev (817), momentskewkurtosis (805)

Listing: ./mathex/ex35.pp

Program Example35;

{ Program to demonstrate the PopnStdDev function. }

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.

25.13.60 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` (808), `mean` (801), `meanandstddev` (802), `stddev` (817), `momentskewkurtosis` (805)

Listing: ./mathex/ex36.pp

Program Example36;

{ Program to demonstrate the PopnVariance function. }

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;

WriteIn ('Max : ',MaxValue(ExArray):8:4);

WriteIn ('Min : ',MinValue(ExArray):8:4);

WriteIn ('Pop. var. : ',PopnVariance(ExArray):8:4);

WriteIn ('Pop. var. (b) : ',PopnVariance(@ExArray[1],100):8:4);

end.

25.13.61 power

Synopsis: Return real power.

Declaration: `function power(base: Float;exponent: Float) : Float`

Visibility: default

Description: `power` raises `base` to the power `power`. This is equivalent to `exp(power*ln(base))`. Therefore `base` should be non-negative.

Errors: None.

See also: `intpower` (794)

Listing: ./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.

```

25.13.62 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` (774)) 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` (791), `InterestRate` (794), `NumberOfPeriods` (807), `Payment` (807), `TPaymentTime` (780), `CashFlowFunctions` (774)

25.13.63 radtcycle

Synopsis: Convert radians angle to cycle angle

Declaration: `function radtcycle(rad: Float) : Float`

Visibility: default

Description: `Radtcycle` 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` (788), `degtorad` (789), `radtodeg` (811), `radtograd` (811), `cycletorad` (787)

Listing: `./mathex/ex37.pp`

Program `Example37`;

{ Program to demonstrate the radtcycle function. }

Uses `math`;

```

begin
  writeln(radtocycle(2*pi):8:6);
  writeln(radtocycle(pi):8:6);
  writeln(radtocycle(pi/2):8:6);
end.

```

25.13.64 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` (788), `degtorad` (789), `radtocycle` (810), `radtograd` (811), `cycletorad` (787)

Listing: `./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.

```

25.13.65 radtograd

Synopsis: Convert radians angle to grads angle

Declaration: `function radtograd(rad: Float) : Float`

Visibility: default

Description: `Radtodeg` converts its argument `rad` (an angle expressed in radians) to an angle in grads. (200 grads equals pi radians)

Errors: None.

See also: `degtograd` (788), `degtorad` (789), `radtocycle` (810), `radtodeg` (811), `cycletorad` (787)

Listing: `./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.

25.13.66 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` ([801](#)), `stddev` ([817](#)), `meanandstddev` ([802](#))

Listing: ./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.

25.13.67 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`

Description: `SameValue` returns `True` if the floating-point values `A` and `B` are the same, i.e. whether the absolute value of their difference is smaller than `Epsilon`. If their difference is larger, then `False` is returned.

If unspecified, the default value for `Epsilon` is 0.0.

See also: `MinFloat` (778), `IsZero` (795)

25.13.71 `sec`

Synopsis: Alias for `secant`

Declaration: `function sec(x: Float) : Float`

Visibility: default

Description: `sec` is an alias for the `secant` (814) function.

See also: `secant` (814)

25.13.72 `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` (786)

25.13.73 `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.

25.13.74 `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.

25.13.75 SetRoundMode

Synopsis: Set the Floating Point Unit rounding mode.

Declaration: `function SetRoundMode(const RoundMode: TFPU RoundingMode)
: TFPU RoundingMode`

Visibility: default

Description: Set the Floating Point Unit rounding mode.

25.13.76 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.

25.13.77 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, but rounds up, and returns the result. This result is accurate to "10 to the power Digits". It uses the standard `Round` function for this.

See also: `TRoundToRange` ([780](#)), `RoundTo` ([813](#))

25.13.78 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` (782), `arccos` (780)

Listing: `./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.

25.13.79 sinh

Synopsis: Return hyperbolic sine

Declaration: `function sinh`(`x`: `Float`) : `Float`

Visibility: default

Description: `Sinh` returns the hyperbolic sine of its argument `x`.

See also: `cosh` (786), `arsinh` (783), `tanh` (821), `artanh` (784)

Listing: `./mathex/ex42.pp`

Program `Example42`;

{ Program to demonstrate the sinh function. }

Uses `math`;

```

begin
  writeln(sinh(0));
  writeln(sinh(1));
  writeln(sinh(-1));
end.

```

25.13.80 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 ([801](#)), meanandstddev ([802](#)), variance ([822](#)), totalvariance ([821](#))

Listing: ./mathex/ex43.pp

Program Example40;

{ Program to demonstrate the stddev function. }

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[0],10000):8:4);

end.

25.13.81 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 \(818\)](#), [sumsandsquares \(819\)](#), [totalvariance \(821\)](#), [variance \(822\)](#)

Listing: ./mathex/ex44.pp

Program Example44;

{ Program to demonstrate the Sum function. }

Uses math;

Var

```
  I : 1..100;
  ExArray : Array[1..100] of Float;
```

begin

```
  Randomize;
```

```
  for I:=low(ExArray) to high(ExArray) do
```

```
    ExArray[I]:=(RandomRandom)*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.
```

25.13.82 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
```

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.

25.13.83 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` (817), `sumsandsquares` (819), `totalvariance` (821), `variance` (822)

Listing: `./mathex/ex45.pp`

Program `Example45;`

{ Program to demonstrate the SumOfSquares function. }

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.

25.13.84 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` (817), `sumofsquares` (818), `totalvariance` (821), `variance` (822)

Listing: ./mathex/ex46.pp

Program Example45;

{ Program to demonstrate the SumOfSquares function. }

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;

WriteIn('Max' : ',MaxValue(ExArray):8:4);

WriteIn('Min' : ',MinValue(ExArray):8:4);

 SumsAndSquares(ExArray,S,SS);

WriteIn('Sum' : ',S:8:4);

WriteIn('Sum squares' : ',SS:8:4);

 SumsAndSquares(@ExArray[1],100,S,SS);

WriteIn('Sum (b)' : ',S:8:4);

WriteIn('Sum squares (b)' : ',SS:8:4);

end.

25.13.85 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 3pi/2 then an overflow will occur.

See also: tanh ([821](#)), arcsin ([782](#)), sincos ([815](#)), arccos ([780](#))

Listing: ./mathex/ex47.pp

Program Example47;

{ Program to demonstrate the Tan function. }

Uses math;

Procedure DoTan(Angle : Float);

begin

Write('Angle' : ',RadToDeg(Angle):8:6);

WriteIn('Tangent' : ',Tan(Angle):8:6);

end;

begin

 DoTan(0);

```

DoTan(Pi);
DoTan(Pi/3);
DoTan(Pi/4);
DoTan(Pi/6);
end.

```

25.13.86 tanh

Synopsis: Return hyperbolic tangent

Declaration: `function tanh(x: Float) : Float`

Visibility: default

Description: Tanh returns the hyperbolic tangent of x.

Errors: None.

See also: [arcsin \(782\)](#), [sincos \(815\)](#), [arccos \(780\)](#)

Listing: ./mathex/ex48.pp

Program Example48;

{ Program to demonstrate the Tanh function. }

Uses math;

```

begin
  writeln(tanh(0));
  writeln(tanh(1));
  writeln(tanh(-1));
end.

```

25.13.87 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 \(822\)](#), [stddev \(817\)](#), [mean \(801\)](#)

Listing: ./mathex/ex49.pp

```

Program Example49;

{ Program to demonstrate the TotalVariance function. }

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);
  WriteIn('Total variance      : ',TV:8:4);
  TV:=TotalVariance(@ExArray[1],100);
  WriteIn('Total Variance (b) : ',TV:8:4);
end.

```

25.13.88 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 (821), stddev (817), mean (801)

Listing: ./mathex/ex50.pp

```

Program Example50;

{ Program to demonstrate the Variance function. }

Uses math;

Var
  I : 1..100;
  ExArray : Array[1..100] of Float;

```

```
V : float;  
  
begin  
  Randomize;  
  for i:=low(ExArray) to high(ExArray) do  
    ExArray[i]:=(Random-Random)*100;  
  V:=Variance(ExArray);  
  Writeln('Variance      : ',V:8:4);  
  V:=Variance(@ExArray[1],100);  
  Writeln('Variance (b) : ',V:8:4);  
end.
```

25.14 EInvalidArgument

25.14.1 Description

Exception raised when invalid arguments are passed to a function.

Chapter 26

Reference for unit 'matrix'

26.1 Overview

The unit `matrix` is a unit that provides objects for the common two, three and four dimensional vectors matrixes. These vectors and matrixes 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 matrixes. The functionality needs of a general matrix unit varies from application to application; one can think of reduced memory usage tricks for matrixes 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.

26.2 Constants, types and variables

26.2.1 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.

26.3 Procedures and functions

26.3.1 operator *(Tmatrix2_double, Double): Tmatrix2_double

Synopsis: Multiply a two-dimensional double precision matrix by a scalar

Declaration: `operator operator *(Tmatrix2_double, Double): Tmatrix2_double`

```
(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.

26.3.2 operator *(Tmatrix2_double, Tmatrix2_double): Tmatrix2_double

Synopsis: Give product of two two-dimensional double precision matrices

Declaration: `operator operator *(Tmatrix2_double, Tmatrix2_double): Tmatrix2_double`

```
(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.

26.3.3 operator*(Tmatrix2_double, Tvector2_double): Tvector2_double

Synopsis: Give product of a two-dimensional double precision matrix and vector

Declaration: `operator operator *(Tmatrix2_double, Tvector2_double): Tvector2_double`

```
(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.

26.3.4 operator*(Tmatrix2_extended, extended): Tmatrix2_extended

Synopsis: Multiply a two-dimensional extended precision matrix by a scalar

Declaration: `operator operator *(Tmatrix2_extended, extended): Tmatrix2_extended`

```
(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.

26.3.5 operator*(Tmatrix2_extended, Tmatrix2_extended): Tmatrix2_extended

Synopsis: Give product of two two-dimensional extended precision matrices

Declaration:

```
operator 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.

26.3.6 operator*(Tmatrix2_extended, Tvector2_extended): Tvector2_extended

Synopsis: Give product of a two-dimensional extended precision matrix and vector

Declaration:

```
operator 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.

26.3.7 operator *(Tmatrix2_single, single): Tmatrix2_single

Synopsis: Multiply a two-dimensional single precision matrix by a scalar

Declaration: `operator operator *(Tmatrix2_single, single): Tmatrix2_single`

```
(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.

26.3.8 operator *(Tmatrix2_single, Tmatrix2_single): Tmatrix2_single

Synopsis: Give product of two two-dimensional single precision matrices

Declaration: `operator operator *(Tmatrix2_single, Tmatrix2_single): Tmatrix2_single`

```
(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.

26.3.9 operator *(Tmatrix2_single, Tvector2_single): Tvector2_single

Synopsis: Give product of a two-dimensional single precision matrix and vector

Declaration: `operator operator *(Tmatrix2_single, Tvector2_single): Tvector2_single`

```
(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.

26.3.10 operator *(Tmatrix3_double, Double): Tmatrix3_double

Synopsis: Multiply a three-dimensional double precision matrix by a scalar

Declaration: `operator operator *(Tmatrix3_double, Double): Tmatrix3_double`

```
(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.

26.3.11 operator*(Tmatrix3_double, Tmatrix3_double): Tmatrix3_double

Synopsis: Give product of two three-dimensional double precision matrices

Declaration: `operator operator *(Tmatrix3_double, Tmatrix3_double): Tmatrix3_double`

```
(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.

26.3.12 operator*(Tmatrix3_double, Tvector3_double): Tvector3_double

Synopsis: Give product of a three-dimensional double precision matrix and vector

Declaration: `operator operator *(Tmatrix3_double, Tvector3_double): Tvector3_double`

```
(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.

26.3.13 operator*(Tmatrix3_extended, extended): Tmatrix3_extended

Synopsis: Multiply a three-dimensional extended precision matrix by a scalar

Declaration: `operator operator *(Tmatrix3_extended, extended): Tmatrix3_extended`

```
(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.

26.3.14 operator*(Tmatrix3_extended, Tmatrix3_extended): Tmatrix3_extended

Synopsis: Give product of two three-dimensional extended precision matrices

Declaration:

```
operator
(const m
const m2
: Tmatr
```

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.

26.3.15 operator*(Tmatrix3_extended, Tvector3_extended): Tvector3_extended

Synopsis: Give product of a three-dimensional extended precision matrix and vector

Declaration:

```
operator
(const m
const v:
: Tvect
```

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.

26.3.16 operator*(Tmatrix3_single, single): Tmatrix3_single

Synopsis: Multiply a three-dimensional single precision matrix by a scalar

Declaration: `operator operator *(Tmatrix3_single, single): Tmatrix3_single`

```
(const m: Tmatrix3_sing
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.

26.3.17 operator*(Tmatrix3_single, Tmatrix3_single): Tmatrix3_single

Synopsis: Give product of two three-dimensional single precision matrices

Declaration: `operator operator *(Tmatrix3_single, Tmatrix3_single): Tmatrix3_single`

```
(const m1: Tma
const m2: Tmat
: Tmatrix3_si
```

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.

26.3.18 operator*(Tmatrix3_single, Tvector3_single): Tvector3_single

Synopsis: Give product of a three-dimensional single precision matrix and vector

Declaration: `operator operator *(Tmatrix3_single, Tvector3_single): Tvector3_single`

```
(const m: Tmat
const v: Tvect
: Tvector3_si
```

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.

26.3.19 operator*(Tmatrix4_double, Double): Tmatrix4_double

Synopsis: Multiply a four-dimensional double precision matrix by a scalar

Declaration: `operator operator *(Tmatrix4_double, Double): Tmatrix4_double`

```
(const m: Tmatrix4_doub
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.

26.3.20 operator*(Tmatrix4_double, Tmatrix4_double): Tmatrix4_double

Synopsis: Give product of two four-dimensional double precision matrices

Declaration: `operator operator *(Tmatrix4_double, Tmatrix4_double): Tmatrix4_double`

```
(const m1: Tmat
const m2: Tmat
: Tmatrix4_do
```

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.

26.3.21 operator*(Tmatrix4_double, Tvector4_double): Tvector4_double

Synopsis: Give product of a four-dimensional double precision matrix and vector

Declaration: `operator operator *(Tmatrix4_double, Tvector4_double): Tvector4_double`

```
(const m: Tmat
const v: Tvect
: Tvector4_do
```

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.

26.3.22 operator*(Tmatrix4_extended, extended): Tmatrix4_extended

Synopsis: Multiply a four-dimensional extended precision matrix by a scalar

Declaration: `operator operator *(Tmatrix4_extended, extended): Tmatrix4_extended`

```
(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.

26.3.23 operator*(Tmatrix4_extended, Tmatrix4_extended): Tmatrix4_extended

Synopsis: Give product of two four-dimensional extended precision matrices

Declaration:

```
operator operator *(Tmatrix4_extended, 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.

26.3.24 operator*(Tmatrix4_extended, Tvector4_extended): Tvector4_extended

Synopsis: Give product of a four-dimendional extended precision matrix and vector

Declaration:

```
operator operator *(Tmatrix4_extended, 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.

26.3.25 operator*(Tmatrix4_single, single): Tmatrix4_single

Synopsis: Multiply a four-dimensional single precision matrix by a scalar

Declaration: `operator operator *(Tmatrix4_single, single): Tmatrix4_single`

```
(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.

26.3.26 operator *(Tmatrix4_single, Tmatrix4_single): Tmatrix4_single

Synopsis: Give product of two four-dimensional single precision matrices

Declaration: `operator operator *(Tmatrix4_single, Tmatrix4_single): Tmatrix4_single`

```
(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.

26.3.27 operator *(Tmatrix4_single, Tvector4_single): Tvector4_single

Synopsis: Give product of a four-dimensional single precision matrix and vector

Declaration: `operator operator *(Tmatrix4_single, Tvector4_single): Tvector4_single`

```
(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.

26.3.28 operator *(Tvector2_double, Double): Tvector2_double

Synopsis: Multiply a two-dimensional double precision vector by a scalar

Declaration: `operator operator *(Tvector2_double, Double): Tvector2_double`

```
(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.

26.3.29 operator *(Tvector2_double, Tvector2_double): Tvector2_double

Synopsis: Multiply two vectors element wise

Declaration: `operator operator *(Tvector2_double, Tvector2_double): Tvector2_double`

```
(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.

26.3.30 operator *(Tvector2_extended, extended): Tvector2_extended

Synopsis: Multiply a two-dimensional extended precision vector by a scalar

Declaration: `operator operator *(Tvector2_extended, extended): Tvector2_extended`

```
(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.

26.3.31 operator *(Tvector2_extended, Tvector2_extended): Tvector2_extended

Synopsis: Multiply two vectors element wise

Declaration:

```
operator operator *(Tvector2_extended, Tvector2_extended): Tvector2_extended
(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.

26.3.32 operator *(Tvector2_single, single): Tvector2_single

Synopsis: Multiply a two-dimensional single precision vector by a scalar

Declaration: `operator operator *(Tvector2_single, single): Tvector2_single`

```
(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.

26.3.33 operator *(Tvector2_single, Tvector2_single): Tvector2_single

Synopsis: Multiply two vectors element wise

Declaration: `operator operator *(Tvector2_single, Tvector2_single): Tvector2_single`

```
(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.

26.3.34 operator *(Tvector3_double, Double): Tvector3_double

Synopsis: Multiply a three-dimensional double precision vector by a scalar

Declaration: `operator operator *(Tvector3_double, Double): Tvector3_double`

```
(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.

26.3.35 operator *(Tvector3_double, Tvector3_double): Tvector3_double

Synopsis: Multiply two vectors element wise

Declaration: `operator operator *(Tvector3_double, Tvector3_double): Tvector3_double`

```
(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.

26.3.36 operator *(Tvector3_extended, extended): Tvector3_extended

Synopsis: Multiply a three-dimensional extended precision vector by a scalar

Declaration: `operator operator *(Tvector3_extended, extended): Tvector3_extended`

```
(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.

26.3.37 operator *(Tvector3_extended, Tvector3_extended): Tvector3_extended

Synopsis: Multiply two vectors element wise

Declaration:

```
operator 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.

26.3.38 operator *(Tvector3_single, single): Tvector3_single

Synopsis: Multiply a three-dimensional single precision vector by a scalar

Declaration: `operator operator *(Tvector3_single, single): Tvector3_single`

```
(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.

26.3.39 operator *(Tvector3_single, Tvector3_single): Tvector3_single

Synopsis: Multiply two vectors element wise

Declaration: `operator operator *(Tvector3_single, Tvector3_single): Tvector3_single`

```
(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.

26.3.40 operator *(Tvector4_double, Double): Tvector4_double

Synopsis: Multiply a four-dimensional double precision vector by a scalar

Declaration: `operator operator *(Tvector4_double, Double): Tvector4_double`

```
(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.

26.3.41 operator *(Tvector4_double, Tvector4_double): Tvector4_double

Synopsis: Multiply two vectors element wise

Declaration: `operator operator *(Tvector4_double, Tvector4_double): Tvector4_double`

```
(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.

26.3.42 operator *(Tvector4_extended, extended): Tvector4_extended

Synopsis: Multiply a four-dimensional extended precision vector by a scalar

Declaration: `operator operator *(Tvector4_extended, extended): Tvector4_extended`

```
(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.

26.3.43 operator *(Tvector4_extended, Tvector4_extended): Tvector4_extended

Synopsis: Multiply two vectors element wise

Declaration:

```
operator operator *(Tvector4_extended, Tvector4_extended): Tvector4_extended
(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.

26.3.44 operator *(Tvector4_single, single): Tvector4_single

Synopsis: Multiply a four-dimensional single precision vector by a scalar

Declaration: `operator operator *(Tvector4_single, single): Tvector4_single`

```
(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.

26.3.45 operator *(Tvector4_single, Tvector4_single): Tvector4_single

Synopsis: Multiply two vectors element wise

Declaration: `operator operator *(Tvector4_single, Tvector4_single): Tvector4_single`

```
(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.

26.3.46 operator **(Tvector2_double, Tvector2_double): Double

Synopsis: Calculate the internal product of two vectors.

Declaration: `operator operator **(Tvector2_double, Tvector2_double): Double`
`(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.

26.3.47 operator **(Tvector2_extended, Tvector2_extended): extended

Synopsis: Calculate the internal product of two vectors.

Declaration: `operator operator **(Tvector2_extended, Tvector2_extended): extended`
`(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.

26.3.48 operator **(Tvector2_single, Tvector2_single): single

Synopsis: Calculate the internal product of two vectors.

Declaration: `operator operator **(Tvector2_single, Tvector2_single): single`
`(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.

26.3.49 operator **(Tvector3_double, Tvector3_double): Double

Synopsis: Calculate the internal product of two vectors.

Declaration: `operator operator **(Tvector3_double, Tvector3_double): Double`
`(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.

26.3.50 operator**(Tvector3_extended, Tvector3_extended): extended

Synopsis: Calculate the internal product of two vectors.

Declaration: `operator operator**(Tvector3_extended, Tvector3_extended): extended`

```
(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.

26.3.51 operator**(Tvector3_single, Tvector3_single): single

Synopsis: Calculate the internal product of two vectors.

Declaration: `operator operator**(Tvector3_single, Tvector3_single): single`

```
(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.

26.3.52 operator**(Tvector4_double, Tvector4_double): Double

Synopsis: Calculate the internal product of two vectors.

Declaration: `operator operator**(Tvector4_double, Tvector4_double): Double`

```
(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.

26.3.53 operator**(Tvector4_extended, Tvector4_extended): extended

Synopsis: Calculate the internal product of two vectors.

Declaration: `operator operator**(Tvector4_extended, Tvector4_extended): extended`

```
(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.

26.3.54 operator ******(Tvector4_single, Tvector4_single): single

Synopsis: Calculate the internal product of two vectors.

Declaration: `operator operator **(Tvector4_single, Tvector4_single): single`

```
(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.

26.3.55 operator **+**(Tmatrix2_double, Double): Tmatrix2_double

Synopsis: Add scalar to two-dimensional double precision matrix

Declaration: `operator operator +(Tmatrix2_double, Double): Tmatrix2_double`

```
(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.

26.3.56 operator **+**(Tmatrix2_double, Tmatrix2_double): Tmatrix2_double

Synopsis: Add two two-dimensional double precision matrices together.

Declaration: `operator operator +(Tmatrix2_double, Tmatrix2_double): Tmatrix2_double`

```
(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.

26.3.57 operator **+**(Tmatrix2_extended, extended): Tmatrix2_extended

Synopsis: Add scalar to two-dimensional extended precision matrix

Declaration: `operator operator +(Tmatrix2_extended, extended): Tmatrix2_extended`

```
(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.

26.3.58 operator +(Tmatrix2_extended, Tmatrix2_extended): Tmatrix2_extended

Synopsis: Add two two-dimensional extended precision matrices together.

Declaration:

```
operator
(const m
const m2
: Tmatr
```

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.

26.3.59 operator +(Tmatrix2_single, single): Tmatrix2_single

Synopsis: Add scalar to two-dimensional single precision matrix

Declaration: operator operator +(Tmatrix2_single, single): Tmatrix2_single

```
(const m: Tmatrix2_sing
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.

26.3.60 operator +(Tmatrix2_single, Tmatrix2_single): Tmatrix2_single

Synopsis: Add two two-dimensional single precision matrices together.

Declaration: operator operator +(Tmatrix2_single, Tmatrix2_single): Tmatrix2_single

```
(const m1: Tma
const m2: Tmat
: Tmatrix2_si
```

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.

26.3.61 operator +(Tmatrix3_double, Double): Tmatrix3_double

Synopsis: Add scalar to three-dimensional double precision matrix

Declaration: operator operator +(Tmatrix3_double, Double): Tmatrix3_double

```
(const m: Tmatrix3_doub
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.

26.3.62 operator +(Tmatrix3_double, Tmatrix3_double): Tmatrix3_double

Synopsis: Add two three-dimensional double precision matrices together.

Declaration: `operator operator +(Tmatrix3_double, Tmatrix3_double): Tmatrix3_double`

```
(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.

26.3.63 operator +(Tmatrix3_extended, extended): Tmatrix3_extended

Synopsis: Add scalar to three-dimensional extended precision matrix

Declaration: `operator operator +(Tmatrix3_extended, extended): Tmatrix3_extended`

```
(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.

26.3.64 operator +(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.

26.3.65 operator +(Tmatrix3_single, single): Tmatrix3_single

Synopsis: Add scalar to three-dimensional single precision matrix

Declaration: `operator operator +(Tmatrix3_single, single): Tmatrix3_single`

```
(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.

26.3.66 operator +(Tmatrix3_single, Tmatrix3_single): Tmatrix3_single

Synopsis: Add two three-dimensional single precision matrices together.

Declaration: `operator operator +(Tmatrix3_single, Tmatrix3_single): Tmatrix3_single`

```
(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.

26.3.67 operator +(Tmatrix4_double, Double): Tmatrix4_double

Synopsis: Add scalar to four-dimensional double precision matrix

Declaration: `operator operator +(Tmatrix4_double, Double): Tmatrix4_double`

```
(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.

26.3.68 operator +(Tmatrix4_double, Tmatrix4_double): Tmatrix4_double

Synopsis: Add two four-dimensional double precision matrices together.

Declaration: `operator operator +(Tmatrix4_double, Tmatrix4_double): Tmatrix4_double`

```
(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.

26.3.69 operator +(Tmatrix4_extended, extended): Tmatrix4_extended

Synopsis: Add scalar to four-dimensional extended precision matrix

Declaration: `operator operator +(Tmatrix4_extended, extended): Tmatrix4_extended`

```
(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.

26.3.70 operator +(Tmatrix4_extended, Tmatrix4_extended): Tmatrix4_extended

Synopsis: Add two four-dimensional extended precision matrices together.

Declaration:

```
operator
(const m
const m2
: Tmatr
```

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.

26.3.71 operator +(Tmatrix4_single, single): Tmatrix4_single

Synopsis: Add scalar to four-dimensional single precision matrix

Declaration: operator operator +(Tmatrix4_single, single): Tmatrix4_single

```
(const m: Tmatrix4_sing
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.

26.3.72 operator +(Tmatrix4_single, Tmatrix4_single): Tmatrix4_single

Synopsis: Add two four-dimensional single precision matrices together.

Declaration: operator operator +(Tmatrix4_single, Tmatrix4_single): Tmatrix4_single

```
(const m1: Tma
const m2: Tmat
: Tmatrix4_si
```

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.

26.3.73 operator +(Tvector2_double, Double): Tvector2_double

Synopsis: Add scalar to two-dimensional double precision vector

Declaration: operator operator +(Tvector2_double, Double): Tvector2_double

```
(const x: Tvector2_doub
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.

26.3.74 operator +(Tvector2_double, Tvector2_double): Tvector2_double

Synopsis: Add two-dimensional double precision vectors together

Declaration: `operator operator +(Tvector2_double, Tvector2_double): Tvector2_double`

```
(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.

26.3.75 operator +(Tvector2_extended, extended): Tvector2_extended

Synopsis: Add scalar to two-dimensional extended precision vector

Declaration: `operator operator +(Tvector2_extended, extended): Tvector2_extended`

```
(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.

26.3.76 operator +(Tvector2_extended, Tvector2_extended): Tvector2_extended

Synopsis: Add two-dimensional extended precision vectors together

Declaration:

```
operator operator +(Tvector2_extended, Tvector2_extended)
(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.

26.3.77 operator +(Tvector2_single, single): Tvector2_single

Synopsis: Add scalar to two-dimensional single precision vector

Declaration: `operator operator +(Tvector2_single, single): Tvector2_single`

```
(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.

26.3.78 operator +(Tvector2_single, Tvector2_single): Tvector2_single

Synopsis: Add two-dimensional single precision vectors together

Declaration: `operator operator +(Tvector2_single, Tvector2_single): Tvector2_single`

```
(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.

26.3.79 operator +(Tvector3_double, Double): Tvector3_double

Synopsis: Add scalar to three-dimensional double precision vector

Declaration: `operator operator +(Tvector3_double, Double): Tvector3_double`

```
(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.

26.3.80 operator +(Tvector3_double, Tvector3_double): Tvector3_double

Synopsis: Add three-dimensional double precision vectors together

Declaration: `operator operator +(Tvector3_double, Tvector3_double): Tvector3_double`

```
(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.

26.3.81 operator +(Tvector3_extended, extended): Tvector3_extended

Synopsis: Add scalar to three-dimensional extended precision vector

Declaration: `operator operator +(Tvector3_extended, extended): Tvector3_extended`

```
(const x: Tvector3_double,
y: extended)
: 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.

26.3.82 operator +(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.

26.3.83 operator +(Tvector3_single, single): Tvector3_single

Synopsis: Add scalar to three-dimensional single precision vector

Declaration: operator operator +(Tvector3_single, single): Tvector3_single

```
(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.

26.3.84 operator +(Tvector3_single, Tvector3_single): Tvector3_single

Synopsis: Add three-dimensional extended precision vectors together

Declaration: operator operator +(Tvector3_single, Tvector3_single): Tvector3_single

```
(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.

26.3.85 operator +(Tvector4_double, Double): Tvector4_double

Synopsis: Add scalar to four-dimensional double precision vector

Declaration: operator operator +(Tvector4_double, Double): Tvector4_double

```
(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.

26.3.86 operator +(Tvector4_double, Tvector4_double): Tvector4_double

Synopsis: Add four-dimensional double precision vectors together

Declaration: `operator operator +(Tvector4_double, Tvector4_double): Tvector4_double`

```
(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.

26.3.87 operator +(Tvector4_extended, extended): Tvector4_extended

Synopsis: Add scalar to four-dimensional extended precision vector

Declaration: `operator operator +(Tvector4_extended, extended): Tvector4_extended`

```
(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.

26.3.88 operator +(Tvector4_extended, Tvector4_extended): Tvector4_extended

Synopsis: Add four-dimensional extended precision vectors together

Declaration:

```
operator 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.

26.3.89 operator +(Tvector4_single, single): Tvector4_single

Synopsis: Add scalar to four-dimensional single precision vector

Declaration: `operator operator +(Tvector4_single, single): Tvector4_single`

```
(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.

26.3.90 operator +(Tvector4_single, Tvector4_single): Tvector4_single

Synopsis: Add four-dimensional single precision vectors together

Declaration: `operator operator +(Tvector4_single, Tvector4_single): Tvector4_single`

```
(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.

26.3.91 operator -(Tmatrix2_double): Tmatrix2_double

Synopsis: Negate two-dimensional double precision matrix.

Declaration: `operator operator -(Tmatrix2_double): Tmatrix2_double`

```
(const m1: Tmatrix2_double)
: Tmatrix2_double
```

Visibility: default

Description: This operation returns a matrix with all elements negated.

26.3.92 operator -(Tmatrix2_double, Double): Tmatrix2_double

Synopsis: Subtract scalar to two-dimensional double precision matrix

Declaration: `operator operator -(Tmatrix2_double, Double): Tmatrix2_double`

```
(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.

26.3.93 operator -(Tmatrix2_double, Tmatrix2_double): Tmatrix2_double

Synopsis: Subtract a two-dimensional double precision matrix from another.

Declaration: `operator operator -(Tmatrix2_double, Tmatrix2_double): Tmatrix2_double`

```
(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.

26.3.94 operator -(Tmatrix2_extended): Tmatrix2_extended

Synopsis: Negate two-dimensional extended precision matrix.

Declaration: `operator operator -(Tmatrix2_extended): Tmatrix2_extended`
`(const m1: Tmatrix2_extended)`
`: Tmatrix2_extended`

Visibility: default

Description: This operation returns a matrix with all elements negated.

26.3.95 operator -(Tmatrix2_extended, extended): Tmatrix2_extended

Synopsis: Add scalar to two-dimensional extended precision matrix

Declaration: `operator operator -(Tmatrix2_extended, extended): Tmatrix2_extended`
`(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.

26.3.96 operator -(Tmatrix2_extended, Tmatrix2_extended): Tmatrix2_extended

Synopsis: Subtract a two-dimensional extended precision matrix from another.

Declaration: `operator operator -(Tmatrix2_extended, Tmatrix2_extended): Tmatrix2_extended`
`(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.

26.3.97 operator -(Tmatrix2_single): Tmatrix2_single

Synopsis: Negate two-dimensional single precision matrix.

Declaration: `operator operator -(Tmatrix2_single): Tmatrix2_single`
`(const m1: Tmatrix2_single)`
`: Tmatrix2_single`

Visibility: default

Description: This operation returns a matrix with all elements negated.

26.3.98 operator -(Tmatrix2_single, single): Tmatrix2_single

Synopsis: Subtract scalar to two-dimensional single precision matrix

Declaration: `operator operator -(Tmatrix2_single, single): Tmatrix2_single`

```
(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.

26.3.99 operator -(Tmatrix2_single, Tmatrix2_single): Tmatrix2_single

Synopsis: Subtract a two-dimensional single precision matrix from another.

Declaration: `operator operator -(Tmatrix2_single, Tmatrix2_single): Tmatrix2_single`

```
(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.

26.3.100 operator -(Tmatrix3_double): Tmatrix3_double

Synopsis: Negate three-dimensional double precision matrix.

Declaration: `operator operator -(Tmatrix3_double): Tmatrix3_double`

```
(const m1: Tmatrix3_double)
: Tmatrix3_double
```

Visibility: default

Description: This operation returns a matrix with all elements negated.

26.3.101 operator -(Tmatrix3_double, Double): Tmatrix3_double

Synopsis: Add scalar to three-dimensional double precision matrix

Declaration: `operator operator -(Tmatrix3_double, Double): Tmatrix3_double`

```
(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.

26.3.102 operator -(Tmatrix3_double, Tmatrix3_double): Tmatrix3_double

Synopsis: Subtract a three-dimensional double precision matrix from another.

Declaration: `operator operator -(Tmatrix3_double, Tmatrix3_double): Tmatrix3_double`

```
(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.

26.3.103 operator -(Tmatrix3_extended): Tmatrix3_extended

Synopsis: Negate three-dimensional extended precision matrix.

Declaration: `operator operator -(Tmatrix3_extended): Tmatrix3_extended`

```
(const m1: Tmatrix3_extended)
: Tmatrix3_extended
```

Visibility: default

Description: This operation returns a matrix with all elements negated.

26.3.104 operator -(Tmatrix3_extended, extended): Tmatrix3_extended

Synopsis: Add scalar to three-dimensional extended precision matrix

Declaration: `operator operator -(Tmatrix3_extended, extended): Tmatrix3_extended`

```
(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.

26.3.105 operator -(Tmatrix3_extended, Tmatrix3_extended): Tmatrix3_extended

Synopsis: Subtract a three-dimensional extended precision matrix from another.

Declaration:

```
operator 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.

26.3.106 operator -(Tmatrix3_single): Tmatrix3_single

Synopsis: Negate three-dimensional single precision matrix.

Declaration: `operator operator -(Tmatrix3_single): Tmatrix3_single`
`(const m1: Tmatrix3_single)`
`: Tmatrix3_single`

Visibility: default

Description: This operation returns a matrix with all elements negated.

26.3.107 operator -(Tmatrix3_single, single): Tmatrix3_single

Synopsis: Add scalar to three-dimensional single precision matrix

Declaration: `operator operator -(Tmatrix3_single, single): Tmatrix3_single`
`(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.

26.3.108 operator -(Tmatrix3_single, Tmatrix3_single): Tmatrix3_single

Synopsis: Subtract a three-dimensional single precision matrix from another.

Declaration: `operator operator -(Tmatrix3_single, Tmatrix3_single): Tmatrix3_single`
`(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.

26.3.109 operator -(Tmatrix4_double): Tmatrix4_double

Synopsis: Negate four-dimensional double precision matrix.

Declaration: `operator operator -(Tmatrix4_double): Tmatrix4_double`
`(const m1: Tmatrix4_double)`
`: Tmatrix4_double`

Visibility: default

Description: This operation returns a matrix with all elements negated.

26.3.110 operator -(Tmatrix4_double, Double): Tmatrix4_double

Synopsis: Add scalar to four-dimensional double precision matrix

Declaration: `operator operator -(Tmatrix4_double, Double): Tmatrix4_double`

```
(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.

26.3.111 operator -(Tmatrix4_double, Tmatrix4_double): Tmatrix4_double

Synopsis: Subtract a four-dimensional double precision matrix from another.

Declaration: `operator operator -(Tmatrix4_double, Tmatrix4_double): Tmatrix4_double`

```
(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.

26.3.112 operator -(Tmatrix4_extended): Tmatrix4_extended

Synopsis: Negate four-dimensional extended precision matrix.

Declaration: `operator operator -(Tmatrix4_extended): Tmatrix4_extended`

```
(const m1: Tmatrix4_extended)
: Tmatrix4_extended
```

Visibility: default

Description: This operation returns a matrix with all elements negated.

26.3.113 operator -(Tmatrix4_extended, extended): Tmatrix4_extended

Synopsis: Add scalar to four-dimensional extended precision matrix

Declaration: `operator operator -(Tmatrix4_extended, extended): Tmatrix4_extended`

```
(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.

26.3.114 operator -(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.

26.3.115 operator -(Tmatrix4_single): Tmatrix4_single

Synopsis: Negate four-dimensional single precision matrix.

Declaration:

```
operator operator -(Tmatrix4_single): Tmatrix4_single
(const m1: Tmatrix4_single)
: Tmatrix4_single
```

Visibility: default

Description: This operation returns a matrix with all elements negated.

26.3.116 operator -(Tmatrix4_single, single): Tmatrix4_single

Synopsis: Add scalar to four-dimensional single precision matrix

Declaration:

```
operator operator -(Tmatrix4_single, single): Tmatrix4_single
(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.

26.3.117 operator -(Tmatrix4_single, Tmatrix4_single): Tmatrix4_single

Synopsis: Subtract a four-dimensional single precision matrix from another.

Declaration:

```
operator operator -(Tmatrix4_single, Tmatrix4_single): Tmatrix4_single
(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.

26.3.118 operator -(Tvector2_double): Tvector2_double

Synopsis: Negate two-dimensional vector.

Declaration: `operator operator -(Tvector2_double) : Tvector2_double`
`(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.

26.3.119 operator -(Tvector2_double, Double): Tvector2_double

Synopsis: Subtract scalar from two-dimensional double precision vector

Declaration: `operator operator -(Tvector2_double, Double) : Tvector2_double`
`(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.

26.3.120 operator -(Tvector2_double, Tvector2_double): Tvector2_double

Synopsis: Subtract two-dimensional double precision vectors from each other

Declaration: `operator operator -(Tvector2_double, Tvector2_double) : Tvector2_double`
`(const x: Tvec`
`const y: Tvect`
`: 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.

26.3.121 operator -(Tvector2_extended): Tvector2_extended

Synopsis: Negate two-dimensional vector.

Declaration: `operator operator -(Tvector2_extended) : Tvector2_extended`
`(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.

26.3.122 operator -(Tvector2_extended, extended): Tvector2_extended

Synopsis: Subtract scalar from two-dimensional extended precision vector

Declaration: `operator operator -(Tvector2_extended, extended): Tvector2_extended`
`(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.

26.3.123 operator -(Tvector2_extended, Tvector2_extended): Tvector2_extended

Synopsis: Subtract two-dimensional extended precision vectors from each other

Declaration: `operator operator -(Tvector2_extended, Tvector2_extended): Tvector2_extended`
`(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.

26.3.124 operator -(Tvector2_single): Tvector2_single

Synopsis: Negate two-dimensional vector.

Declaration: `operator operator -(Tvector2_single): Tvector2_single`
`(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.

26.3.125 operator -(Tvector2_single, single): Tvector2_single

Synopsis: Subtract scalar from two-dimensional single precision vector

Declaration: `operator operator -(Tvector2_single, single): Tvector2_single`
`(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.

26.3.126 operator -(Tvector2_single, Tvector2_single): Tvector2_single

Synopsis: Subtract two-dimensional single precision vectors from each other

Declaration: `operator operator -(Tvector2_single, Tvector2_single): Tvector2_single`

```
(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.

26.3.127 operator -(Tvector3_double): Tvector3_double

Synopsis: Negate three-dimensional vector.

Declaration: `operator operator -(Tvector3_double): Tvector3_double`

```
(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.

26.3.128 operator -(Tvector3_double, Double): Tvector3_double

Synopsis: Subtract scalar from three-dimensional double precision vector

Declaration: `operator operator -(Tvector3_double, Double): Tvector3_double`

```
(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.

26.3.129 operator -(Tvector3_double, Tvector3_double): Tvector3_double

Synopsis: Subtract three-dimensional double precision vectors from each other

Declaration: `operator operator -(Tvector3_double, Tvector3_double): Tvector3_double`

```
(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.

26.3.130 operator -(Tvector3_extended): Tvector3_extended

Synopsis: Negate three-dimensional vector.

Declaration: `operator operator -(Tvector3_extended) : Tvector3_extended`
`(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.

26.3.131 operator -(Tvector3_extended, extended): Tvector3_extended

Synopsis: Subtract scalar from three-dimensional extended precision vector

Declaration: `operator operator -(Tvector3_extended, extended) : Tvector3_extended`
`(const x: Tvector`
`y: extended)`
`: Tvector3_exten`

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.

26.3.132 operator -(Tvector3_extended, Tvector3_extended): Tvector3_extended

Synopsis: Subtract three-dimensional extended precision vectors from each other

Declaration: `operator operator -(Tvector3_extended, Tvector3_extended) : Tvector3_extended`
`operator`
`(const x`
`const y:`
`: Tvect`

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.

26.3.133 operator -(Tvector3_single): Tvector3_single

Synopsis: Negate three-dimensional vector.

Declaration: `operator operator -(Tvector3_single) : Tvector3_single`
`(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.

26.3.134 operator -(Tvector3_single, single): Tvector3_single

Synopsis: Subtract scalar from three-dimensional single precision vector

Declaration: `operator operator -(Tvector3_single, single): Tvector3_single`
`(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.

26.3.135 operator -(Tvector3_single, Tvector3_single): Tvector3_single

Synopsis: Subtract three-dimensional single precision vectors from each other

Declaration: `operator operator -(Tvector3_single, Tvector3_single): Tvector3_single`
`(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.

26.3.136 operator -(Tvector4_double): Tvector4_double

Synopsis: Negate four-dimensional vector.

Declaration: `operator operator -(Tvector4_double): Tvector4_double`
`(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.

26.3.137 operator -(Tvector4_double, Double): Tvector4_double

Synopsis: Subtract scalar from four-dimensional double precision vector

Declaration: `operator operator -(Tvector4_double, Double): Tvector4_double`
`(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.

26.3.138 operator -(Tvector4_double, Tvector4_double): Tvector4_double

Synopsis: Subtract four-dimensional double precision vectors from each other

Declaration: `operator operator -(Tvector4_double, Tvector4_double): Tvector4_double`

```
(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.

26.3.139 operator -(Tvector4_extended): Tvector4_extended

Synopsis: Negate four-dimensional vector.

Declaration: `operator operator -(Tvector4_extended): Tvector4_extended`

```
(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.

26.3.140 operator -(Tvector4_extended, extended): Tvector4_extended

Synopsis: Subtract scalar from four-dimensional extended precision vector

Declaration: `operator operator -(Tvector4_extended, extended): Tvector4_extended`

```
(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.

26.3.141 operator -(Tvector4_extended, Tvector4_extended): Tvector4_extended

Synopsis: Subtract four-dimensional extended precision vectors from each other

Declaration:

```
operator 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.

26.3.142 operator -(Tvector4_single): Tvector4_single

Synopsis: Negate four-dimensional vector.

Declaration: `operator operator -(Tvector4_single): Tvector4_single`
`(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.

26.3.143 operator -(Tvector4_single, single): Tvector4_single

Synopsis: Subtract scalar from four-dimensional single precision vector

Declaration: `operator operator -(Tvector4_single, single): Tvector4_single`
`(const x: Tvector4_sing`
`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.

26.3.144 operator -(Tvector4_single, Tvector4_single): Tvector4_single

Synopsis: Subtract four-dimensional single precision vectors from each other

Declaration: `operator operator -(Tvector4_single, Tvector4_single): Tvector4_single`
`(const x: Tvec`
`const y: Tvect`
`: Tvector4_si`

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.

26.3.145 operator /(Tmatrix2_double, Double): Tmatrix2_double

Synopsis: Divide a two-dimensional single precision matrix by a scalar

Declaration: `operator operator /(Tmatrix2_double, Double): Tmatrix2_double`
`(const m: Tmatrix2_doub`
`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.

26.3.146 operator /(Tmatrix2_extended, extended): Tmatrix2_extended

Synopsis: Divide a two-dimensional single precision matrix by a scalar

Declaration: `operator operator / (Tmatrix2_extended, extended): Tmatrix2_extended`

```
(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.

26.3.147 operator /(Tmatrix2_single, single): Tmatrix2_single

Synopsis: Divide a two-dimensional single precision matrix by a scalar

Declaration: `operator operator / (Tmatrix2_single, single): Tmatrix2_single`

```
(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.

26.3.148 operator /(Tmatrix3_double, Double): Tmatrix3_double

Synopsis: Divide a two-dimensional single precision matrix by a scalar

Declaration: `operator operator / (Tmatrix3_double, Double): Tmatrix3_double`

```
(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.

26.3.149 operator /(Tmatrix3_extended, extended): Tmatrix3_extended

Synopsis: Divide a two-dimensional single precision matrix by a scalar

Declaration: `operator operator / (Tmatrix3_extended, extended): Tmatrix3_extended`

```
(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.

26.3.150 operator /(Tmatrix3_single, single): Tmatrix3_single

Synopsis: Divide a two-dimensional single precision matrix by a scalar

Declaration: `operator operator / (Tmatrix3_single, single): Tmatrix3_single`

```
(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.

26.3.151 operator /(Tmatrix4_double, Double): Tmatrix4_double

Synopsis: Divide a two-dimensional single precision matrix by a scalar

Declaration: `operator operator / (Tmatrix4_double, Double): Tmatrix4_double`

```
(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.

26.3.152 operator /(Tmatrix4_extended, extended): Tmatrix4_extended

Synopsis: Divide a two-dimensional single precision matrix by a scalar

Declaration: `operator operator / (Tmatrix4_extended, extended): Tmatrix4_extended`

```
(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.

26.3.153 operator /(Tmatrix4_single, single): Tmatrix4_single

Synopsis: Divide a two-dimensional single precision matrix by a scalar

Declaration: `operator operator / (Tmatrix4_single, single): Tmatrix4_single`

```
(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.

26.3.154 operator /(Tvector2_double, Double): Tvector2_double

Synopsis: Divide a two-dimensional double precision vector by a scalar

Declaration: `operator operator / (Tvector2_double, Double): Tvector2_double`
`(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.

26.3.155 operator /(Tvector2_extended, extended): Tvector2_extended

Synopsis: Divide a two-dimensional extended precision vector by a scalar

Declaration: `operator operator / (Tvector2_extended, extended): Tvector2_extended`
`(const x: Tvector`
`y: extended)`
`: Tvector2_exten`

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.

26.3.156 operator /(Tvector2_single, single): Tvector2_single

Synopsis: Divide a two-dimensional single precision vector by a scalar

Declaration: `operator operator / (Tvector2_single, single): Tvector2_single`
`(const x: Tvector2_sing`
`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.

26.3.157 operator /(Tvector3_double, Double): Tvector3_double

Synopsis: Divide a three-dimensional double precision vector by a scalar

Declaration: `operator operator / (Tvector3_double, Double): Tvector3_double`
`(const x: Tvector3_doub`
`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.

26.3.158 operator /(Tvector3_extended, extended): Tvector3_extended

Synopsis: Divide a three-dimensional extended precision vector by a scalar

Declaration: `operator operator / (Tvector3_extended, extended) : Tvector3_extended`

```
(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.

26.3.159 operator /(Tvector3_single, single): Tvector3_single

Synopsis: Divide a three-dimensional single precision vector by a scalar

Declaration: `operator operator / (Tvector3_single, single) : Tvector3_single`

```
(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.

26.3.160 operator /(Tvector4_double, Double): Tvector4_double

Synopsis: Divide a four-dimensional double precision vector by a scalar

Declaration: `operator operator / (Tvector4_double, Double) : Tvector4_double`

```
(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.

26.3.161 operator /(Tvector4_extended, extended): Tvector4_extended

Synopsis: Divide a four-dimensional extended precision vector by a scalar

Declaration: `operator operator / (Tvector4_extended, extended) : Tvector4_extended`

```
(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.

26.3.162 operator /(Tvector4_single, single): Tvector4_single

Synopsis: Divide a four-dimensional single precision vector by a scalar

Declaration: `operator operator /(Tvector4_single, single): Tvector4_single`

```
(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.

26.3.163 operator :=(Tmatrix2_double): Tmatrix2_extended

Synopsis: Allow assignment of two-dimensional double precision matrix to two-dimensional extended precision matrix

Declaration: `operator operator :=(Tmatrix2_double): Tmatrix2_extended`

```
(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.

26.3.164 operator :=(Tmatrix2_double): Tmatrix2_single

Synopsis: Allow assignment of two-dimensional double precision matrix to two-dimensional single precision matrix

Declaration: `operator operator :=(Tmatrix2_double): Tmatrix2_single`

```
(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.

26.3.165 operator :=(Tmatrix2_double): Tmatrix3_double

Synopsis: Allow assignment of two-dimensional double precision matrix to three-dimensional double precision matrix

Declaration: `operator operator :=(Tmatrix2_double): Tmatrix3_double`

```
(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.

26.3.166 operator :=(Tmatrix2_double): Tmatrix3_extended

Synopsis: Allow assignment of two-dimensional double precision matrix to three-dimensional extended precision matrix

Declaration: `operator operator :=(Tmatrix2_double): Tmatrix3_extended`
`(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.

26.3.167 operator :=(Tmatrix2_double): Tmatrix3_single

Synopsis: Allow assignment of two-dimensional single precision matrix to three-dimensional single precision matrix

Declaration: `operator operator :=(Tmatrix2_double): Tmatrix3_single`
`(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.

26.3.168 operator :=(Tmatrix2_double): Tmatrix4_double

Synopsis: Allow assignment of two-dimensional double precision matrix to four-dimensional double precision matrix

Declaration: `operator operator :=(Tmatrix2_double): Tmatrix4_double`
`(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.

26.3.169 operator :=(Tmatrix2_double): Tmatrix4_extended

Synopsis: Allow assignment of two-dimensional double precision matrix to four-dimensional extended precision matrix

Declaration: `operator operator :=(Tmatrix2_double): Tmatrix4_extended`
`(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.

26.3.170 operator :=(Tmatrix2_double): Tmatrix4_single

Synopsis: Allow assignment of two-dimensional double precision matrix to four-dimensional single precision matrix

Declaration: `operator operator :=(Tmatrix2_double): Tmatrix4_single`
`(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.

26.3.171 operator :=(Tmatrix2_extended): Tmatrix2_double

Synopsis: Allow assignment of two-dimensional extended precision matrix to two-dimensional double precision matrix

Declaration: `operator operator :=(Tmatrix2_extended): Tmatrix2_double`
`(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.

26.3.172 operator :=(Tmatrix2_extended): Tmatrix2_single

Synopsis: Allow assignment of two-dimensional extended precision matrix to two-dimensional single precision matrix

Declaration: `operator operator :=(Tmatrix2_extended): Tmatrix2_single`
`(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.

26.3.173 operator :=(Tmatrix2_extended): Tmatrix3_double

Synopsis: Allow assignment of two-dimensional extended precision matrix to three-dimensional double precision matrix

Declaration: `operator operator :=(Tmatrix2_extended): Tmatrix3_double`
`(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.

26.3.174 operator :=(Tmatrix2_extended): Tmatrix3_extended

Synopsis: Allow assignment of two-dimensional extended precision matrix to three-dimensional extended precision matrix

Declaration: `operator operator :=(Tmatrix2_extended): Tmatrix3_extended`
`(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.

26.3.175 operator :=(Tmatrix2_extended): Tmatrix3_single

Synopsis: Allow assignment of two-dimensional extended precision matrix to three-dimensional single precision matrix

Declaration: `operator operator :=(Tmatrix2_extended): Tmatrix3_single`
`(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.

26.3.176 operator :=(Tmatrix2_extended): Tmatrix4_double

Synopsis: Allow assignment of two-dimensional extended precision matrix to four-dimensional double precision matrix

Declaration: `operator operator :=(Tmatrix2_extended): Tmatrix4_double`
`(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.

26.3.177 operator :=(Tmatrix2_extended): Tmatrix4_extended

Synopsis: Allow assignment of two-dimensional extended precision matrix to four-dimensional extended precision matrix

Declaration: `operator operator :=(Tmatrix2_extended): Tmatrix4_extended`
`(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.

26.3.178 operator :=(Tmatrix2_extended): Tmatrix4_single

Synopsis: Allow assignment of two-dimensional extended precision matrix to four-dimensional single precision matrix

Declaration: `operator operator :=(Tmatrix2_extended): Tmatrix4_single`
`(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.

26.3.179 operator :=(Tmatrix2_single): Tmatrix2_double

Synopsis: Allow assignment of two-dimensional single precision matrix to two-dimensional double precision matrix

Declaration: `operator operator :=(Tmatrix2_single): Tmatrix2_double`
`(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.

26.3.180 operator :=(Tmatrix2_single): Tmatrix2_extended

Synopsis: Allow assignment of two-dimensional single precision matrix to two-dimensional extended precision matrix

Declaration: `operator operator :=(Tmatrix2_single): Tmatrix2_extended`
`(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.

26.3.181 operator :=(Tmatrix2_single): Tmatrix3_double

Synopsis: Allow assignment of two-dimensional single precision matrix to three-dimensional double precision matrix

Declaration: `operator operator :=(Tmatrix2_single): Tmatrix3_double`
`(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.

26.3.182 operator :=(Tmatrix2_single): Tmatrix3_extended

Synopsis: Allow assignment of two-dimensional single precision matrix to three-dimensional extended precision matrix

Declaration: `operator operator :=(Tmatrix2_single): Tmatrix3_extended`
`(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.

26.3.183 operator :=(Tmatrix2_single): Tmatrix3_single

Synopsis: Allow assignment of two-dimensional single precision matrix to three-dimensional single precision matrix

Declaration: `operator operator :=(Tmatrix2_single): Tmatrix3_single`
`(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.

26.3.184 operator :=(Tmatrix2_single): Tmatrix4_double

Synopsis: Allow assignment of two-dimensional single precision matrix to four-dimensional double precision matrix

Declaration: `operator operator :=(Tmatrix2_single): Tmatrix4_double`
`(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.

26.3.185 operator :=(Tmatrix2_single): Tmatrix4_extended

Synopsis: Allow assignment of two-dimensional single precision matrix to four-dimensional extended precision matrix

Declaration: `operator operator :=(Tmatrix2_single): Tmatrix4_extended`
`(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.

26.3.186 operator :=(Tmatrix2_single): Tmatrix4_single

Synopsis: Allow assignment of two-dimensional single precision matrix to four-dimensional single precision matrix

Declaration: `operator operator :=(Tmatrix2_single): Tmatrix4_single`
`(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.

26.3.187 operator :=(Tmatrix3_double): Tmatrix2_double

Synopsis: Allow assignment of three-dimensional double precision matrix to two-dimensional double precision matrix

Declaration: `operator operator :=(Tmatrix3_double): Tmatrix2_double`
`(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.

26.3.188 operator :=(Tmatrix3_double): Tmatrix2_extended

Synopsis: Allow assignment of three-dimensional double precision matrix to two-dimensional extended precision matrix

Declaration: `operator operator :=(Tmatrix3_double): Tmatrix2_extended`
`(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.

26.3.189 operator :=(Tmatrix3_double): Tmatrix2_single

Synopsis: Allow assignment of three-dimensional double precision matrix to two-dimensional single precision matrix

Declaration: `operator operator :=(Tmatrix3_double): Tmatrix2_single`
`(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.

26.3.190 operator :=(Tmatrix3_double): Tmatrix3_extended

Synopsis: Allow assignment of three-dimensional double precision matrix to three-dimensional extended precision matrix

Declaration: `operator operator :=(Tmatrix3_double): Tmatrix3_extended`
`(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.

26.3.191 operator :=(Tmatrix3_double): Tmatrix3_single

Synopsis: Allow assignment of three-dimensional double precision matrix to three-dimensional single precision matrix

Declaration: `operator operator :=(Tmatrix3_double): Tmatrix3_single`
`(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.

26.3.192 operator :=(Tmatrix3_double): Tmatrix4_double

Synopsis: Allow assignment of three-dimensional double precision matrix to four-dimensional double precision matrix

Declaration: `operator operator :=(Tmatrix3_double): Tmatrix4_double`
`(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.

26.3.193 operator :=(Tmatrix3_double): Tmatrix4_extended

Synopsis: Allow assignment of three-dimensional double precision matrix to four-dimensional extended precision matrix

Declaration: `operator operator := (Tmatrix3_double) : Tmatrix4_extended`
`(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.

26.3.194 operator :=(Tmatrix3_double): Tmatrix4_single

Synopsis: Allow assignment of three-dimensional double precision matrix to four-dimensional single precision matrix

Declaration: `operator operator := (Tmatrix3_double) : Tmatrix4_single`
`(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.

26.3.195 operator :=(Tmatrix3_extended): Tmatrix2_double

Synopsis: Allow assignment of three-dimensional extended precision matrix to two-dimensional double precision matrix

Declaration: `operator operator := (Tmatrix3_extended) : Tmatrix2_double`
`(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.

26.3.196 operator :=(Tmatrix3_extended): Tmatrix2_extended

Synopsis: Allow assignment of three-dimensional extended precision matrix to two-dimensional extended precision matrix

Declaration: `operator operator := (Tmatrix3_extended) : Tmatrix2_extended`
`(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.

26.3.197 operator :=(Tmatrix3_extended): Tmatrix2_single

Synopsis: Allow assignment of three-dimensional extended precision matrix to two-dimensional single precision matrix

Declaration: `operator operator :=(Tmatrix3_extended): Tmatrix2_single`
`(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.

26.3.198 operator :=(Tmatrix3_extended): Tmatrix3_double

Synopsis: Allow assignment of three-dimensional extended precision matrix to three-dimensional double precision matrix

Declaration: `operator operator :=(Tmatrix3_extended): Tmatrix3_double`
`(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.

26.3.199 operator :=(Tmatrix3_extended): Tmatrix3_single

Synopsis: Allow assignment of three-dimensional extended precision matrix to three-dimensional single precision matrix

Declaration: `operator operator :=(Tmatrix3_extended): Tmatrix3_single`
`(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.

26.3.200 operator :=(Tmatrix3_extended): Tmatrix4_double

Synopsis: Allow assignment of three-dimensional extended precision matrix to four-dimensional double precision matrix

Declaration: `operator operator :=(Tmatrix3_extended): Tmatrix4_double`
`(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.

26.3.201 operator :=(Tmatrix3_extended): Tmatrix4_extended

Synopsis: Allow assignment of three-dimensional extended precision matrix to four-dimensional extended precision matrix

Declaration: `operator operator :=(Tmatrix3_extended): Tmatrix4_extended`
`(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.

26.3.202 operator :=(Tmatrix3_extended): Tmatrix4_single

Synopsis: Allow assignment of three-dimensional extended precision matrix to four-dimensional single precision matrix

Declaration: `operator operator :=(Tmatrix3_extended): Tmatrix4_single`
`(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.

26.3.203 operator :=(Tmatrix3_single): Tmatrix2_double

Synopsis: Allow assignment of three-dimensional single precision matrix to two-dimensional double precision matrix

Declaration: `operator operator :=(Tmatrix3_single): Tmatrix2_double`
`(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.

26.3.204 operator :=(Tmatrix3_single): Tmatrix2_extended

Synopsis: Allow assignment of three-dimensional single precision matrix to two-dimensional extended precision matrix

Declaration: `operator operator :=(Tmatrix3_single): Tmatrix2_extended`
`(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.

26.3.205 operator :=(Tmatrix3_single): Tmatrix2_single

Synopsis: Allow assignment of three-dimensional single precision matrix to two-dimensional single precision matrix

Declaration: `operator operator :=(Tmatrix3_single): Tmatrix2_single`
`(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.

26.3.206 operator :=(Tmatrix3_single): Tmatrix3_double

Synopsis: Allow assignment of three-dimensional single precision matrix to three-dimensional double precision matrix

Declaration: `operator operator :=(Tmatrix3_single): Tmatrix3_double`
`(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.

26.3.207 operator :=(Tmatrix3_single): Tmatrix3_extended

Synopsis: Allow assignment of three-dimensional single precision matrix to three-dimensional extended precision matrix

Declaration: `operator operator :=(Tmatrix3_single): Tmatrix3_extended`
`(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.

26.3.208 operator :=(Tmatrix3_single): Tmatrix4_double

Synopsis: Allow assignment of three-dimensional single precision matrix to four-dimensional double precision matrix

Declaration: `operator operator :=(Tmatrix3_single): Tmatrix4_double`
`(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.

26.3.209 operator :=(Tmatrix3_single): Tmatrix4_extended

Synopsis: Allow assignment of three-dimensional single precision matrix to four-dimensional extended precision matrix

Declaration: `operator operator :=(Tmatrix3_single): Tmatrix4_extended`
`(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.

26.3.210 operator :=(Tmatrix3_single): Tmatrix4_single

Synopsis: Allow assignment of three-dimensional single precision matrix to four-dimensional single precision matrix

Declaration: `operator operator :=(Tmatrix3_single): Tmatrix4_single`
`(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.

26.3.211 operator :=(Tmatrix4_double): Tmatrix2_double

Synopsis: Allow assignment of four-dimensional double precision matrix to two-dimensional double precision matrix

Declaration: `operator operator :=(Tmatrix4_double): Tmatrix2_double`
`(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.

26.3.212 operator :=(Tmatrix4_double): Tmatrix2_extended

Synopsis: Allow assignment of four-dimensional double precision matrix to two-dimensional extended precision matrix

Declaration: `operator operator :=(Tmatrix4_double): Tmatrix2_extended`
`(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.

26.3.213 operator :=(Tmatrix4_double): Tmatrix2_single

Synopsis: Allow assignment of four-dimensional double precision matrix to two-dimensional single precision matrix

Declaration: `operator operator :=(Tmatrix4_double): Tmatrix2_single`
`(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.

26.3.214 operator :=(Tmatrix4_double): Tmatrix3_double

Synopsis: Allow assignment of four-dimensional double precision matrix to three-dimensional double precision matrix

Declaration: `operator operator :=(Tmatrix4_double): Tmatrix3_double`
`(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.

26.3.215 operator :=(Tmatrix4_double): Tmatrix3_extended

Synopsis: Allow assignment of four-dimensional double precision matrix to three-dimensional extended precision matrix

Declaration: `operator operator :=(Tmatrix4_double): Tmatrix3_extended`
`(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.

26.3.216 operator :=(Tmatrix4_double): Tmatrix3_single

Synopsis: Allow assignment of four-dimensional double precision matrix to three-dimensional single precision matrix

Declaration: `operator operator :=(Tmatrix4_double): Tmatrix3_single`
`(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.

26.3.217 operator :=(Tmatrix4_double): Tmatrix4_extended

Synopsis: Allow assignment of four-dimensional double precision matrix to four-dimensional extended precision matrix

Declaration: `operator operator :=(Tmatrix4_double): Tmatrix4_extended`
`(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.

26.3.218 operator :=(Tmatrix4_double): Tmatrix4_single

Synopsis: Allow assignment of four-dimensional single precision matrix to four-dimensional single precision matrix

Declaration: `operator operator :=(Tmatrix4_double): Tmatrix4_single`
`(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.

26.3.219 operator :=(Tmatrix4_extended): Tmatrix2_double

Synopsis: Allow assignment of four-dimensional extended precision matrix to two-dimensional double precision matrix

Declaration: `operator operator :=(Tmatrix4_extended): Tmatrix2_double`
`(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.

26.3.220 operator :=(Tmatrix4_extended): Tmatrix2_extended

Synopsis: Allow assignment of four-dimensional extended precision matrix to two-dimensional extended precision matrix

Declaration: `operator operator :=(Tmatrix4_extended): Tmatrix2_extended`
`(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.

26.3.221 operator :=(Tmatrix4_extended): Tmatrix2_single

Synopsis: Allow assignment of four-dimensional extended precision matrix to two-dimensional single precision matrix

Declaration: `operator operator :=(Tmatrix4_extended): Tmatrix2_single`
`(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.

26.3.222 operator :=(Tmatrix4_extended): Tmatrix3_double

Synopsis: Allow assignment of four-dimensional extended precision matrix to three-dimensional double precision matrix

Declaration: `operator operator :=(Tmatrix4_extended): Tmatrix3_double`
`(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.

26.3.223 operator :=(Tmatrix4_extended): Tmatrix3_extended

Synopsis: Allow assignment of four-dimensional extended precision matrix to three-dimensional extended precision matrix

Declaration: `operator operator :=(Tmatrix4_extended): Tmatrix3_extended`
`(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.

26.3.224 operator :=(Tmatrix4_extended): Tmatrix3_single

Synopsis: Allow assignment of four-dimensional extended precision matrix to three-dimensional single precision matrix

Declaration: `operator operator :=(Tmatrix4_extended): Tmatrix3_single`
`(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.

26.3.225 operator :=(Tmatrix4_extended): Tmatrix4_double

Synopsis: Allow assignment of four-dimensional extended precision matrix to four-dimensional double precision matrix

Declaration: `operator operator :=(Tmatrix4_extended): Tmatrix4_double`
`(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.

26.3.226 operator :=(Tmatrix4_extended): Tmatrix4_single

Synopsis: Allow assignment of four-dimensional extended precision matrix to four-dimensional single precision matrix

Declaration: `operator operator :=(Tmatrix4_extended): Tmatrix4_single`
`(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.

26.3.227 operator :=(Tmatrix4_single): Tmatrix2_double

Synopsis: Allow assignment of four-dimensional single precision matrix to two-dimensional double precision matrix

Declaration: `operator operator :=(Tmatrix4_single): Tmatrix2_double`
`(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.

26.3.228 operator :=(Tmatrix4_single): Tmatrix2_extended

Synopsis: Allow assignment of four-dimensional single precision matrix to two-dimensional extended precision matrix

Declaration: `operator operator :=(Tmatrix4_single): Tmatrix2_extended`
`(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.

26.3.229 operator :=(Tmatrix4_single): Tmatrix2_single

Synopsis: Allow assignment of four-dimensional single precision matrix to two-dimensional single precision matrix

Declaration: `operator operator :=(Tmatrix4_single): Tmatrix2_single`
`(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.

26.3.230 operator :=(Tmatrix4_single): Tmatrix3_double

Synopsis: Allow assignment of four-dimensional single precision matrix to three-dimensional double precision matrix

Declaration: `operator operator :=(Tmatrix4_single): Tmatrix3_double`
`(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.

26.3.231 operator :=(Tmatrix4_single): Tmatrix3_extended

Synopsis: Allow assignment of four-dimensional single precision matrix to three-dimensional extended precision matrix

Declaration: `operator operator :=(Tmatrix4_single): Tmatrix3_extended`
`(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.

26.3.232 operator :=(Tmatrix4_single): Tmatrix3_single

Synopsis: Allow assignment of four-dimensional single precision matrix to three-dimensional single precision matrix

Declaration: `operator operator :=(Tmatrix4_single): Tmatrix3_single`
`(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.

26.3.233 operator :=(Tmatrix4_single): Tmatrix4_double

Synopsis: Allow assignment of four-dimensional single precision matrix to four-dimensional double precision matrix

Declaration: `operator operator :=(Tmatrix4_single): Tmatrix4_double`
`(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.

26.3.234 operator :=(Tmatrix4_single): Tmatrix4_extended

Synopsis: Allow assignment of four-dimensional single precision matrix to four-dimensional extended precision matrix

Declaration: `operator operator :=(Tmatrix4_single): Tmatrix4_extended`
`(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.

26.3.235 operator :=(Tvector2_double): Tvector2_extended

Synopsis: Allow assignment of double precision vector to extended precision vector

Declaration: `operator operator :=(Tvector2_double): Tvector2_extended`
`(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.

26.3.236 operator :=(Tvector2_double): Tvector2_single

Synopsis: Allow assignment of double precision vector to single precision vector

Declaration: `operator operator :=(Tvector2_double): Tvector2_single`
`(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 losing some precision.

26.3.237 operator :=(Tvector2_double): Tvector3_double

Synopsis: Allow assignment of two-dimensional double precision vector to three-dimensional double precision vector

Declaration: `operator operator :=(Tvector2_double) : Tvector3_double`
`(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.

26.3.238 operator :=(Tvector2_double): Tvector3_extended

Synopsis: Allow assignment of two-dimensional double precision vector to three-dimensional extended precision vector

Declaration: `operator operator :=(Tvector2_double) : Tvector3_extended`
`(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.

26.3.239 operator :=(Tvector2_double): Tvector3_single

Synopsis: Allow assignment of two-dimensional double precision vector to three-dimensional single precision vector

Declaration: `operator operator :=(Tvector2_double) : Tvector3_single`
`(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.

26.3.240 operator :=(Tvector2_double): Tvector4_double

Synopsis: Allow assignment of two-dimensional double precision vector to four-dimensional double precision vector

Declaration: `operator operator :=(Tvector2_double) : Tvector4_double`
`(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.

26.3.241 operator :=(Tvector2_double): Tvector4_extended

Synopsis: Allow assignment of two-dimensional double precision vector to four-dimensional extended precision vector

Declaration: `operator operator :=(Tvector2_double): Tvector4_extended`
`(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.

26.3.242 operator :=(Tvector2_double): Tvector4_single

Synopsis: Allow assignment of two-dimensional double precision vector to four-dimensional single precision vector

Declaration: `operator operator :=(Tvector2_double): Tvector4_single`
`(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.

26.3.243 operator :=(Tvector2_extended): Tvector2_double

Synopsis: Allow assignment of extended precision vector to double precision vector

Declaration: `operator operator :=(Tvector2_extended): Tvector2_double`
`(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 losing some precision.

26.3.244 operator :=(Tvector2_extended): Tvector2_single

Synopsis: Allow assignment of extended precision vector to single precision vector

Declaration: `operator operator :=(Tvector2_extended): Tvector2_single`
`(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 losing some precision.

26.3.245 operator :=(Tvector2_extended): Tvector3_double

Synopsis: Allow assignment of two-dimensional extended precision vector to three-dimensional double precision vector

Declaration: `operator operator :=(Tvector2_extended) : Tvector3_double`
`(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.

26.3.246 operator :=(Tvector2_extended): Tvector3_extended

Synopsis: Allow assignment of two-dimensional extended precision vector to three-dimensional extended precision vector

Declaration: `operator operator :=(Tvector2_extended) : Tvector3_extended`
`(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.

26.3.247 operator :=(Tvector2_extended): Tvector3_single

Synopsis: Allow assignment of two-dimensional extended precision vector to three-dimensional single precision vector

Declaration: `operator operator :=(Tvector2_extended) : Tvector3_single`
`(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.

26.3.248 operator :=(Tvector2_extended): Tvector4_double

Synopsis: Allow assignment of two-dimensional extended precision vector to four-dimensional double precision vector

Declaration: `operator operator :=(Tvector2_extended) : Tvector4_double`
`(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.

26.3.249 operator :=(Tvector2_extended): Tvector4_extended

Synopsis: Allow assignment of two-dimensional extended precision vector to four-dimensional extended precision vector

Declaration: `operator operator :=(Tvector2_extended) : Tvector4_extended`
`(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.

26.3.250 operator :=(Tvector2_extended): Tvector4_single

Synopsis: Allow assignment of two-dimensional extended precision vector to four-dimensional single precision vector

Declaration: `operator operator :=(Tvector2_extended) : Tvector4_single`
`(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.

26.3.251 operator :=(Tvector2_single): Tvector2_double

Synopsis: Allow assignment of single precision vector to double precision vector

Declaration: `operator operator :=(Tvector2_single) : Tvector2_double`
`(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.

26.3.252 operator :=(Tvector2_single): Tvector2_extended

Synopsis: Allow assignment of single precision vector to extended precision vector

Declaration: `operator operator :=(Tvector2_single) : Tvector2_extended`
`(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.

26.3.253 operator :=(Tvector2_single): Tvector3_double

Synopsis: Allow assignment of two-dimensional single precision vector to three-dimensional double precision vector

Declaration: `operator operator :=(Tvector2_single): Tvector3_double`
`(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.

26.3.254 operator :=(Tvector2_single): Tvector3_extended

Synopsis: Allow assignment of two-dimensional single precision vector to three-dimensional extended precision vector

Declaration: `operator operator :=(Tvector2_single): Tvector3_extended`
`(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.

26.3.255 operator :=(Tvector2_single): Tvector3_single

Synopsis: Allow assignment of two-dimensional single precision vector to three-dimensional single precision vector

Declaration: `operator operator :=(Tvector2_single): Tvector3_single`
`(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.

26.3.256 operator :=(Tvector2_single): Tvector4_double

Synopsis: Allow assignment of two-dimensional single precision vector to four-dimensional double precision vector

Declaration: `operator operator :=(Tvector2_single): Tvector4_double`
`(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.

26.3.257 operator :=(Tvector2_single): Tvector4_extended

Synopsis: Allow assignment of two-dimensional single precision vector to four-dimensional extended precision vector

Declaration: `operator operator :=(Tvector2_single): Tvector4_extended`
`(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.

26.3.258 operator :=(Tvector2_single): Tvector4_single

Synopsis: Allow assignment of two-dimensional single precision vector to four-dimensional single precision vector

Declaration: `operator operator :=(Tvector2_single): Tvector4_single`
`(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.

26.3.259 operator :=(Tvector3_double): Tvector2_double

Synopsis: Allow assignment of three-dimensional double precision vector to two-dimensional double precision vector

Declaration: `operator operator :=(Tvector3_double): Tvector2_double`
`(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.

26.3.260 operator :=(Tvector3_double): Tvector2_extended

Synopsis: Allow assignment of three-dimensional double precision vector to two-dimensional extended precision vector

Declaration: `operator operator :=(Tvector3_double): Tvector2_extended`
`(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.

26.3.261 operator :=(Tvector3_double): Tvector2_single

Synopsis: Allow assignment of three-dimensional double precision vector to two-dimensional single precision vector

Declaration: `operator operator :=(Tvector3_double): Tvector2_single`
`(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.

26.3.262 operator :=(Tvector3_double): Tvector3_extended

Synopsis: Allow assignment of three-dimensional double precision vector to three-dimensional extended precision vector

Declaration: `operator operator :=(Tvector3_double): Tvector3_extended`
`(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.

26.3.263 operator :=(Tvector3_double): Tvector3_single

Synopsis: Allow assignment of three-dimensional double precision vector to three-dimensional single precision vector

Declaration: `operator operator :=(Tvector3_double): Tvector3_single`
`(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.

26.3.264 operator :=(Tvector3_double): Tvector4_double

Synopsis: Allow assignment of three-dimensional double precision vector to four-dimensional double precision vector

Declaration: `operator operator :=(Tvector3_double): Tvector4_double`
`(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.

26.3.265 operator :=(Tvector3_double): Tvector4_extended

Synopsis: Allow assignment of three-dimensional double precision vector to four-dimensional extended precision vector

Declaration: `operator operator :=(Tvector3_double): Tvector4_extended`
`(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.

26.3.266 operator :=(Tvector3_double): Tvector4_single

Synopsis: Allow assignment of three-dimensional double precision vector to four-dimensional single precision vector

Declaration: `operator operator :=(Tvector3_double): Tvector4_single`
`(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.

26.3.267 operator :=(Tvector3_extended): Tvector2_double

Synopsis: Allow assignment of three-dimensional extended precision vector to two-dimensional double precision vector

Declaration: `operator operator :=(Tvector3_extended): Tvector2_double`
`(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.

26.3.268 operator :=(Tvector3_extended): Tvector2_extended

Synopsis: Allow assignment of three-dimensional extended precision vector to two-dimensional extended precision vector

Declaration: `operator operator :=(Tvector3_extended): Tvector2_extended`
`(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.

26.3.269 operator :=(Tvector3_extended): Tvector2_single

Synopsis: Allow assignment of three-dimensional extended precision vector to two-dimensional single precision vector

Declaration: `operator operator :=(Tvector3_extended) : Tvector2_single`
`(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.

26.3.270 operator :=(Tvector3_extended): Tvector3_double

Synopsis: Allow assignment of three-dimensional extended precision vector to three-dimensional double precision vector

Declaration: `operator operator :=(Tvector3_extended) : Tvector3_double`
`(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.

26.3.271 operator :=(Tvector3_extended): Tvector3_single

Synopsis: Allow assignment of three-dimensional single precision vector to three-dimensional double precision vector

Declaration: `operator operator :=(Tvector3_extended) : Tvector3_single`
`(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.

26.3.272 operator :=(Tvector3_extended): Tvector4_double

Synopsis: Allow assignment of three-dimensional extended precision vector to four-dimensional double precision vector

Declaration: `operator operator :=(Tvector3_extended) : Tvector4_double`
`(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.

26.3.273 operator :=(Tvector3_extended): Tvector4_extended

Synopsis: Allow assignment of three-dimensional extended precision vector to four-dimensional extended precision vector

Declaration: `operator operator :=(Tvector3_extended): Tvector4_extended`
`(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.

26.3.274 operator :=(Tvector3_extended): Tvector4_single

Synopsis: Allow assignment of three-dimensional extended precision vector to four-dimensional single precision vector

Declaration: `operator operator :=(Tvector3_extended): Tvector4_single`
`(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.

26.3.275 operator :=(Tvector3_single): Tvector2_double

Synopsis: Allow assignment of three-dimensional single precision vector to two-dimensional double precision vector

Declaration: `operator operator :=(Tvector3_single): Tvector2_double`
`(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.

26.3.276 operator :=(Tvector3_single): Tvector2_extended

Synopsis: Allow assignment of three-dimensional single precision vector to two-dimensional extended precision vector

Declaration: `operator operator :=(Tvector3_single): Tvector2_extended`
`(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.

26.3.277 operator :=(Tvector3_single): Tvector2_single

Synopsis: Allow assignment of three-dimensional single precision vector to two-dimensional single precision vector

Declaration: `operator operator :=(Tvector3_single): Tvector2_single`
`(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.

26.3.278 operator :=(Tvector3_single): Tvector3_double

Synopsis: Allow assignment of three-dimensional single precision vector to three-dimensional double precision vector

Declaration: `operator operator :=(Tvector3_single): Tvector3_double`
`(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.

26.3.279 operator :=(Tvector3_single): Tvector3_extended

Synopsis: Allow assignment of three-dimensional single precision vector to three-dimensional extended precision vector

Declaration: `operator operator :=(Tvector3_single): Tvector3_extended`
`(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.

26.3.280 operator :=(Tvector3_single): Tvector4_double

Synopsis: Allow assignment of three-dimensional single precision vector to four-dimensional double precision vector

Declaration: `operator operator :=(Tvector3_single): Tvector4_double`
`(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.

26.3.281 operator :=(Tvector3_single): Tvector4_extended

Synopsis: Allow assignment of three-dimensional single precision vector to four-dimensional extended precision vector

Declaration: `operator operator :=(Tvector3_single): Tvector4_extended`
`(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.

26.3.282 operator :=(Tvector3_single): Tvector4_single

Synopsis: Allow assignment of three-dimensional single precision vector to four-dimensional single precision vector

Declaration: `operator operator :=(Tvector3_single): Tvector4_single`
`(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.

26.3.283 operator :=(Tvector4_double): Tvector2_double

Synopsis: Allow assignment of four-dimensional double precision vector to two-dimensional double precision vector

Declaration: `operator operator :=(Tvector4_double): Tvector2_double`
`(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.

26.3.284 operator :=(Tvector4_double): Tvector2_extended

Synopsis: Allow assignment of four-dimensional double precision vector to two-dimensional extended precision vector

Declaration: `operator operator :=(Tvector4_double): Tvector2_extended`
`(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.

26.3.285 operator :=(Tvector4_double): Tvector2_single

Synopsis: Allow assignment of four-dimensional double precision vector to two-dimensional single precision vector

Declaration: `operator operator :=(Tvector4_double): Tvector2_single`
`(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.

26.3.286 operator :=(Tvector4_double): Tvector3_double

Synopsis: Allow assignment of four-dimensional double precision vector to three-dimensional double precision vector

Declaration: `operator operator :=(Tvector4_double): Tvector3_double`
`(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.

26.3.287 operator :=(Tvector4_double): Tvector3_extended

Synopsis: Allow assignment of four-dimensional double precision vector to three-dimensional extended precision vector

Declaration: `operator operator :=(Tvector4_double): Tvector3_extended`
`(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.

26.3.288 operator :=(Tvector4_double): Tvector3_single

Synopsis: Allow assignment of four-dimensional double precision vector to three-dimensional single precision vector

Declaration: `operator operator :=(Tvector4_double): Tvector3_single`
`(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.

26.3.289 operator :=(Tvector4_double): Tvector4_extended

Synopsis: Allow assignment of four-dimensional single precision vector to four-dimensional extended precision vector

Declaration: `operator operator :=(Tvector4_double): Tvector4_extended`
`(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.

26.3.290 operator :=(Tvector4_double): Tvector4_single

Synopsis: Allow assignment of four-dimensional double precision vector to four-dimensional single precision vector

Declaration: `operator operator :=(Tvector4_double): Tvector4_single`
`(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.

26.3.291 operator :=(Tvector4_extended): Tvector2_double

Synopsis: Allow assignment of four-dimensional extended precision vector to two-dimensional double precision vector

Declaration: `operator operator :=(Tvector4_extended): Tvector2_double`
`(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.

26.3.292 operator :=(Tvector4_extended): Tvector2_extended

Synopsis: Allow assignment of four-dimensional extended precision vector to two-dimensional extended precision vector

Declaration: `operator operator :=(Tvector4_extended) : Tvector2_extended`
`(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.

26.3.293 operator :=(Tvector4_extended): Tvector2_single

Synopsis: Allow assignment of four-dimensional extended precision vector to two-dimensional single precision vector

Declaration: `operator operator :=(Tvector4_extended) : Tvector2_single`
`(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.

26.3.294 operator :=(Tvector4_extended): Tvector3_double

Synopsis: Allow assignment of four-dimensional extended precision vector to three-dimensional double precision vector

Declaration: `operator operator :=(Tvector4_extended) : Tvector3_double`
`(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.

26.3.295 operator :=(Tvector4_extended): Tvector3_extended

Synopsis: Allow assignment of four-dimensional extended precision vector to three-dimensional extended precision vector

Declaration: `operator operator :=(Tvector4_extended) : Tvector3_extended`
`(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.

26.3.296 operator :=(Tvector4_extended): Tvector3_single

Synopsis: Allow assignment of four-dimensional extended precision vector to three-dimensional single precision vector

Declaration: `operator operator :=(Tvector4_extended): Tvector3_single`
`(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.

26.3.297 operator :=(Tvector4_extended): Tvector4_double

Synopsis: Allow assignment of four-dimensional single precision vector to four-dimensional double precision vector

Declaration: `operator operator :=(Tvector4_extended): Tvector4_double`
`(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.

26.3.298 operator :=(Tvector4_extended): Tvector4_single

Synopsis: Allow assignment of four-dimensional extended precision vector to four-dimensional single precision vector

Declaration: `operator operator :=(Tvector4_extended): Tvector4_single`
`(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.

26.3.299 operator :=(Tvector4_single): Tvector2_double

Synopsis: Allow assignment of four-dimensional single precision vector to two-dimensional double precision vector

Declaration: `operator operator :=(Tvector4_single): Tvector2_double`
`(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.

26.3.300 operator :=(Tvector4_single): Tvector2_extended

Synopsis: Allow assignment of four-dimensional single precision vector to two-dimensional extended precision vector

Declaration: `operator operator :=(Tvector4_single): Tvector2_extended`
`(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.

26.3.301 operator :=(Tvector4_single): Tvector2_single

Synopsis: Allow assignment of four-dimensional single precision vector to two-dimensional single precision vector

Declaration: `operator operator :=(Tvector4_single): Tvector2_single`
`(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.

26.3.302 operator :=(Tvector4_single): Tvector3_double

Synopsis: Allow assignment of four-dimensional single precision vector to three-dimensional double precision vector

Declaration: `operator operator :=(Tvector4_single): Tvector3_double`
`(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.

26.3.303 operator :=(Tvector4_single): Tvector3_extended

Synopsis: Allow assignment of four-dimensional single precision vector to three-dimensional extended precision vector

Declaration: `operator operator :=(Tvector4_single): Tvector3_extended`
`(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.

26.3.304 operator :=(Tvector4_single): Tvector3_single

Synopsis: Allow assignment of four-dimensional single precision vector to three-dimensional single precision vector

Declaration: `operator operator :=(Tvector4_single): Tvector3_single`
`(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.

26.3.305 operator :=(Tvector4_single): Tvector4_double

Synopsis: Allow assignment of four-dimensional single precision vector to four-dimensional double precision vector

Declaration: `operator operator :=(Tvector4_single): Tvector4_double`
`(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.

26.3.306 operator :=(Tvector4_single): Tvector4_extended

Synopsis: Allow assignment of four-dimensional single precision vector to four-dimensional extended precision vector

Declaration: `operator operator :=(Tvector4_single): Tvector4_extended`
`(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.

26.3.307 operator ><(Tvector3_double, Tvector3_double): Tvector3_double

Synopsis: Calculate the external product of two three-dimensional vectors

Declaration: `operator operator ><(Tvector3_double, Tvector3_double): Tvector3_double`

```
(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.

26.3.308 operator ><(Tvector3_extended, Tvector3_extended): Tvector3_extended

Synopsis: Calculate the external product of two three-dimensional vectors

Declaration:

```
operator operator ><(Tvector3_extended,
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.

26.3.309 operator ><(Tvector3_single, Tvector3_single): Tvector3_single

Synopsis: Calculate the external product of two three-dimensional vectors

Declaration: `operator operator ><(Tvector3_single, Tvector3_single): Tvector3_single`

```
(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.

26.4 Tmatrix2_double

26.4.1 Description

The `Tmatrix2_double` object provides a matrix of 2*2 double precision scalars.

26.4.2 Method overview

Page	Property	Description
907	<code>determinant</code>	Calculates the determinant of the matrix.
907	<code>get_column</code>	Returns the c-th column of the matrix as vector.
907	<code>get_row</code>	Returns the r-th row of the matrix as vector.
906	<code>init</code>	Initializes the matrix, setting its elements to the values passed to the constructor.
906	<code>init_identity</code>	Initializes the matrix and sets its elements to the identity matrix.
906	<code>init_zero</code>	Initializes the matrix and sets its elements to zero
907	<code>inverse</code>	Calculates the inverse of the matrix.
907	<code>set_column</code>	Sets c-th column of the matrix with a vector.
907	<code>set_row</code>	Sets r-th row of the matrix with a vector.
908	<code>transpose</code>	Returns the transposition of the matrix.

26.4.3 Tmatrix2_double.init_zero

Synopsis: Initializes the matrix and sets its elements to zero

Declaration: `constructor init_zero`

Visibility: default

26.4.4 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.

26.4.5 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.

26.4.6 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.

26.4.7 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.

26.4.8 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.

26.4.9 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.

26.4.10 Tmatrix2_double.determinant

Synopsis: Calculates the determinant of the matrix.

Declaration: `function determinant : Double`

Visibility: default

Description: Returns the determinant of the matrix.

26.4.11 Tmatrix2_double.inverse

Synopsis: Calculates the inverse of the matrix.

Declaration: `function inverse(A_determinant: 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.

26.4.12 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.

26.5 Tmatrix2_extended

26.5.1 Description

The `Tmatrix2_extended` object provides a matrix of 2*2 extended precision scalars.

26.5.2 Method overview

Page	Property	Description
909	<code>determinant</code>	Calculates the determinant of the matrix.
909	<code>get_column</code>	Returns the c-th column of the matrix as vector.
909	<code>get_row</code>	Returns the r-th row of the matrix as vector.
909	<code>init</code>	Initializes the matrix, setting its elements to the values passed to the constructor.
908	<code>init_identity</code>	Initializes the matrix and sets its elements to the identity matrix.
908	<code>init_zero</code>	Initializes the matrix and sets its elements to zero
910	<code>inverse</code>	Calculates the inverse of the matrix.
909	<code>set_column</code>	Sets c-th column of the matrix with a vector.
909	<code>set_row</code>	Sets r-th row of the matrix with a vector.
910	<code>transpose</code>	Returns the transposition of the matrix.

26.5.3 Tmatrix2_extended.init_zero

Synopsis: Initializes the matrix and sets its elements to zero

Declaration: `constructor init_zero`

Visibility: default

26.5.4 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.

26.5.5 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.

26.5.6 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.

26.5.7 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.

26.5.8 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.

26.5.9 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.

26.5.10 Tmatrix2_extended.determinant

Synopsis: Calculates the determinant of the matrix.

Declaration: `function determinant : extended`

Visibility: default

Description: Returns the determinant of the matrix.

26.5.11 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.

26.5.12 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.

26.6 Tmatrix2_single

26.6.1 Description

The Tmatrix2_single object provides a matrix of 2*2 single precision scalars.

26.6.2 Method overview

Page	Property	Description
912	determinant	Calculates the determinant of the matrix.
911	get_column	Returns the c-th column of the matrix as vector.
911	get_row	Returns the r-th row of the matrix as vector.
911	init	Initializes the matrix, setting its elements to the values passed to the constructor.
911	init_identity	Initializes the matrix and sets its elements to the identity matrix.
910	init_zero	Initializes the matrix and sets its elements to zero
912	inverse	Calculates the inverse of the matrix.
911	set_column	Sets c-th column of the matrix with a vector.
912	set_row	Sets r-th row of the matrix with a vector.
912	transpose	Returns the transposition of the matrix.

26.6.3 Tmatrix2_single.init_zero

Synopsis: Initializes the matrix and sets its elements to zero

Declaration: `constructor init_zero`

Visibility: default

26.6.4 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.

26.6.5 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.

26.6.6 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.

26.6.7 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.

26.6.8 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.

26.6.9 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.

26.6.10 Tmatrix2_single.determinant

Synopsis: Calculates the determinant of the matrix.

Declaration: `function determinant : single`

Visibility: default

Description: Returns the determinant of the matrix.

26.6.11 Tmatrix2_single.inverse

Synopsis: Calculates the inverse of the matrix.

Declaration: `function inverse(A_determinant: 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.

26.6.12 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.

26.7 Tmatrix3_double

26.7.1 Description

The `Tmatrix3_double` object provides a matrix of 3*3 double precision scalars.

26.7.2 Method overview

Page	Property	Description
914	determinant	Calculates the determinant of the matrix.
913	get_column	Returns the c-th column of the matrix as vector.
914	get_row	Returns the r-th row of the matrix as vector.
913	init	Initializes the matrix, setting its elements to the values passed to the constructor.
913	init_identity	Initializes the matrix and sets its elements to the identity matrix.
913	init_zero	Initializes the matrix and sets its elements to zero
914	inverse	Calculates the inverse of the matrix.
914	set_column	Sets c-th column of the matrix with a vector.
914	set_row	Sets r-th row of the matrix with a vector.
914	transpose	Returns the transposition of the matrix.

26.7.3 Tmatrix3_double.init_zero

Synopsis: Initializes the matrix and sets its elements to zero

Declaration: `constructor init_zero`

Visibility: default

26.7.4 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.

26.7.5 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.

26.7.6 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.

26.7.7 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.

26.7.8 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.

26.7.9 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.

26.7.10 Tmatrix3_double.determinant

Synopsis: Calculates the determinant of the matrix.

Declaration: `function determinant : Double`

Visibility: default

Description: Returns the determinant of the matrix.

26.7.11 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.

26.7.12 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.

26.8 Tmatrix3_extended

26.8.1 Description

The Tmatrix3_extended object provides a matrix of 3*3 extended precision scalars.

26.8.2 Method overview

Page	Property	Description
916	determinant	Calculates the determinant of the matrix.
916	get_column	Returns the c-th column of the matrix as vector.
916	get_row	Returns the r-th row of the matrix as vector.
915	init	Initializes the matrix, setting its elements to the values passed to the constructor.
915	init_identity	Initializes the matrix and sets its elements to the identity matrix.
915	init_zero	Initializes the matrix and sets its elements to zero
916	inverse	Calculates the inverse of the matrix.
916	set_column	Sets r-th column of the matrix with a vector.
916	set_row	Sets r-th row of the matrix with a vector.
917	transpose	Returns the transposition of the matrix.

26.8.3 Tmatrix3_extended.init_zero

Synopsis: Initializes the matrix and sets its elements to zero

Declaration: `constructor init_zero`

Visibility: default

26.8.4 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.

26.8.5 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.

26.8.6 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.

26.8.7 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.

26.8.8 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.

26.8.9 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.

26.8.10 Tmatrix3_extended.determinant

Synopsis: Calculates the determinant of the matrix.

Declaration: `function determinant : extended`

Visibility: default

Description: Returns the determinant of the matrix.

26.8.11 Tmatrix3_extended.inverse

Synopsis: Calculates the inverse of the matrix.

Declaration: `function inverse(A: Tmatrix3_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.

26.8.12 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.

26.9 Tmatrix3_single

26.9.1 Description

The `Tmatrix3_single` object provides a matrix of 3*3 single precision scalars.

26.9.2 Method overview

Page	Property	Description
919	<code>determinant</code>	Calculates the determinant of the matrix.
918	<code>get_column</code>	Returns the c-th column of the matrix as vector.
918	<code>get_row</code>	Returns the r-th row of the matrix as vector.
918	<code>init</code>	Initializes the matrix, setting its elements to the values passed to the constructor.
917	<code>init_identity</code>	Initializes the matrix and sets its elements to the identity matrix.
917	<code>init_zero</code>	Initializes the matrix and sets its elements to zero
919	<code>inverse</code>	Calculates the inverse of the matrix.
918	<code>set_column</code>	Sets c-th column of the matrix with a vector.
918	<code>set_row</code>	Sets r-th row of the matrix with a vector.
919	<code>transpose</code>	Returns the transposition of the matrix.

26.9.3 Tmatrix3_single.init_zero

Synopsis: Initializes the matrix and sets its elements to zero

Declaration: `constructor init_zero`

Visibility: default

26.9.4 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.

26.9.5 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.

26.9.6 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.

26.9.7 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.

26.9.8 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.

26.9.9 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.

26.9.10 Tmatrix3_single.determinant

Synopsis: Calculates the determinant of the matrix.

Declaration: `function determinant : single`

Visibility: default

Description: Returns the determinant of the matrix.

26.9.11 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.

26.9.12 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.

26.10 Tmatrix4_double

26.10.1 Description

The `Tmatrix4_double` object provides a matrix of 4*4 double precision scalars.

26.10.2 Method overview

Page	Property	Description
921	<code>determinant</code>	Calculates the determinant of the matrix.
920	<code>get_column</code>	Returns the c-th column of the matrix as vector.
920	<code>get_row</code>	Returns the r-th row of the matrix as vector.
920	<code>init</code>	Initializes the matrix, setting its elements to the values passed to the constructor.
920	<code>init_identity</code>	Initializes the matrix and sets its elements to the identity matrix.
920	<code>init_zero</code>	Initializes the matrix and sets its elements to zero
921	<code>inverse</code>	Calculates the inverse of the matrix.
921	<code>set_column</code>	Sets c-th column of the matrix with a vector.
921	<code>set_row</code>	Sets r-th row of the matrix with a vector.
921	<code>transpose</code>	Returns the transposition of the matrix.

26.10.3 Tmatrix4_double.init_zero

Synopsis: Initializes the matrix and sets its elements to zero

Declaration: `constructor init_zero`

Visibility: default

26.10.4 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.

26.10.5 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.

26.10.6 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.

26.10.7 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.

26.10.8 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.

26.10.9 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.

26.10.10 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.

26.10.11 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.

26.10.12 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.

26.11 Tmatrix4_extended

26.11.1 Description

The Tmatrix4_extended object provides a matrix of 4*4 extended precision scalars.

26.11.2 Method overview

Page	Property	Description
923	determinant	Calculates the determinant of the matrix.
923	get_column	Returns the c-th column of the matrix as vector.
923	get_row	Returns the r-th row of the matrix as vector.
922	init	Initializes the matrix, setting its elements to the values passed to the constructor.
922	init_identity	Initializes the matrix and sets its elements to the identity matrix.
922	init_zero	Initializes the matrix and sets its elements to zero
924	inverse	Calculates the inverse of the matrix.
923	set_column	Sets c-th column of the matrix with a vector.
923	set_row	Sets r-th row of the matrix with a vector.
924	transpose	Returns the transposition of the matrix.

26.11.3 Tmatrix4_extended.init_zero

Synopsis: Initializes the matrix and sets its elements to zero

Declaration: `constructor init_zero`

Visibility: default

26.11.4 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.

26.11.5 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.

26.11.6 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.

26.11.7 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.

26.11.8 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.

26.11.9 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.

26.11.10 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.

26.11.11 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.

26.11.12 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.

26.12 Tmatrix4_single

26.12.1 Description

The `Tmatrix4_single` object provides a matrix of 4*4 single precision scalars.

26.12.2 Method overview

Page	Property	Description
926	<code>determinant</code>	Calculates the determinant of the matrix.
925	<code>get_column</code>	Returns the c-th column of the matrix as vector.
925	<code>get_row</code>	Returns the r-th row of the matrix as vector.
925	<code>init</code>	Initializes the matrix, setting its elements to the values passed to the constructor.
925	<code>init_identity</code>	Initializes the matrix and sets its elements to the identity matrix.
924	<code>init_zero</code>	Initializes the matrix and sets its elements to zero
926	<code>inverse</code>	Calculates the inverse of the matrix.
925	<code>set_column</code>	Sets c-th column of the matrix with a vector.
926	<code>set_row</code>	Sets r-th row of the matrix with a vector.
926	<code>transpose</code>	Returns the transposition of the matrix.

26.12.3 Tmatrix4_single.init_zero

Synopsis: Initializes the matrix and sets its elements to zero

Declaration: `constructor init_zero`

Visibility: default

26.12.4 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.

26.12.5 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.

26.12.6 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.

26.12.7 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.

26.12.8 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.

26.12.9 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.

26.12.10 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.

26.12.11 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.

26.12.12 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.

26.13 Tvector2_double

26.13.1 Description

The `Tvector2_double` object provides a vector of two double precision scalars.

26.13.2 Method overview

Page	Property	Description
927	<code>init</code>	Initializes the vector, setting its elements to the values passed to the constructor.
927	<code>init_one</code>	Initializes the vector and sets its elements to one
927	<code>init_zero</code>	Initializes the vector and sets its elements to zero
927	<code>length</code>	Calculates the length of the vector.
927	<code>squared_length</code>	Calculates the squared length of the vector.

26.13.3 Tvector2_double.init_zero

Synopsis: Initializes the vector and sets its elements to zero

Declaration: `constructor init_zero`

Visibility: default

26.13.4 Tvector2_double.init_one

Synopsis: Initializes the vector and sets its elements to one

Declaration: `constructor init_one`

Visibility: default

26.13.5 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

26.13.6 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` ([927](#)) if you are able to, as it is faster.

26.13.7 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`.

26.14 Tvector2_extended

26.14.1 Description

The `Tvector2_extended` object provides a vector of two extended precision scalars.

26.14.2 Method overview

Page	Property	Description
928	<code>init</code>	Initializes the vector, setting its elements to the values passed to the constructor.
928	<code>init_one</code>	Initializes the vector and sets its elements to one
928	<code>init_zero</code>	Initializes the vector and sets its elements to zero
928	<code>length</code>	Calculates the length of the vector.
929	<code>squared_length</code>	Calculates the squared length of the vector.

26.14.3 Tvector2_extended.init_zero

Synopsis: Initializes the vector and sets its elements to zero

Declaration: `constructor init_zero`

Visibility: default

26.14.4 Tvector2_extended.init_one

Synopsis: Initializes the vector and sets its elements to one

Declaration: `constructor init_one`

Visibility: default

26.14.5 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

26.14.6 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` ([929](#)) if you are able to, as it is faster.

26.14.7 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`.

26.15 Tvector2_single

26.15.1 Description

The `Tvector2_single` object provides a vector of two single precision scalars.

26.15.2 Method overview

Page	Property	Description
929	<code>init</code>	Initializes the vector, setting its elements to the values passed to the constructor.
929	<code>init_one</code>	Initializes the vector and sets its elements to one
929	<code>init_zero</code>	Initializes the vector and sets its elements to zero
930	<code>length</code>	Calculates the length of the vector.
930	<code>squared_length</code>	Calculates the squared length of the vector.

26.15.3 Tvector2_single.init_zero

Synopsis: Initializes the vector and sets its elements to zero

Declaration: `constructor init_zero`

Visibility: default

26.15.4 Tvector2_single.init_one

Synopsis: Initializes the vector and sets its elements to one

Declaration: `constructor init_one`

Visibility: default

26.15.5 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

26.15.6 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` (930) if you are able to, as it is faster.

26.15.7 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`.

26.16 Tvector3_double

26.16.1 Description

The `Tvector3_double` object provides a vector of three double precision scalars.

26.16.2 Method overview

Page	Property	Description
931	<code>init</code>	Initializes the vector, setting its elements to the values passed to the constructor.
930	<code>init_one</code>	Initializes the vector and sets its elements to one
930	<code>init_zero</code>	Initializes the vector and sets its elements to zero
931	<code>length</code>	Calculates the length of the vector.
931	<code>squared_length</code>	Calculates the squared length of the vector.

26.16.3 Tvector3_double.init_zero

Synopsis: Initializes the vector and sets its elements to zero

Declaration: `constructor init_zero`

Visibility: default

26.16.4 Tvector3_double.init_one

Synopsis: Initializes the vector and sets its elements to one

Declaration: `constructor init_one`

Visibility: default

26.16.5 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

26.16.6 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` (931) if you are able to, as it is faster.

26.16.7 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`.

26.17 Tvector3_extended

26.17.1 Description

The `Tvector3_extended` object provides a vector of three extended precision scalars.

26.17.2 Method overview

Page	Property	Description
932	<code>init</code>	Initializes the vector, setting its elements to the values passed to the constructor.
932	<code>init_one</code>	Initializes the vector and sets its elements to one
931	<code>init_zero</code>	Initializes the vector and sets its elements to zero
932	<code>length</code>	Calculates the length of the vector.
932	<code>squared_length</code>	Calculates the squared length of the vector.

26.17.3 Tvector3_extended.init_zero

Synopsis: Initializes the vector and sets its elements to zero

Declaration: `constructor init_zero`

Visibility: default

26.17.4 Tvector3_extended.init_one

Synopsis: Initializes the vector and sets its elements to one

Declaration: `constructor init_one`

Visibility: default

26.17.5 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

26.17.6 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` (932) if you are able to, as it is faster.

26.17.7 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`.

26.18 Tvector3_single

26.18.1 Description

The `Tvector3_single` object provides a vector of three single precision scalars.

26.18.2 Method overview

Page	Property	Description
933	<code>init</code>	Initializes the vector, setting its elements to the values passed to the constructor.
933	<code>init_one</code>	Initializes the vector and sets its elements to one
933	<code>init_zero</code>	Initializes the vector and sets its elements to zero
933	<code>length</code>	Calculates the length of the vector.
933	<code>squared_length</code>	Calculates the squared length of the vector.

26.18.3 Tvector3_single.init_zero

Synopsis: Initializes the vector and sets its elements to zero

Declaration: `constructor init_zero`

Visibility: default

26.18.4 Tvector3_single.init_one

Synopsis: Initializes the vector and sets its elements to one

Declaration: `constructor init_one`

Visibility: default

26.18.5 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

26.18.6 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` (933) if you are able to, as it is faster.

26.18.7 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: $\text{squared_length} = \text{data}[0]**2 + \text{data}[1]**2 + \text{data}[2]**2$.

26.19 Tvector4_double

26.19.1 Description

The `Tvector4_double` object provides a vector of four double precision scalars.

26.19.2 Method overview

Page	Property	Description
934	<code>init</code>	Initializes the vector, setting its elements to the values passed to the constructor.
934	<code>init_one</code>	Initializes the vector and sets its elements to one
934	<code>init_zero</code>	Initializes the vector and sets its elements to zero
934	<code>length</code>	Calculates the length of the vector.
934	<code>squared_length</code>	Calculates the squared length of the vector.

26.19.3 Tvector4_double.init_zero

Synopsis: Initializes the vector and sets its elements to zero

Declaration: `constructor init_zero`

Visibility: default

26.19.4 Tvector4_double.init_one

Synopsis: Initializes the vector and sets its elements to one

Declaration: `constructor init_one`

Visibility: default

26.19.5 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

26.19.6 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` ([934](#)) if you are able to, as it is faster.

26.19.7 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`.

26.20 Tvector4_extended

26.20.1 Description

The `Tvector4_extended` object provides a vector of four extended precision scalars.

26.20.2 Method overview

Page	Property	Description
935	<code>init</code>	Initializes the vector, setting its elements to the values passed to the constructor.
935	<code>init_one</code>	Initializes the vector and sets its elements to one
935	<code>init_zero</code>	Initializes the vector and sets its elements to zero
935	<code>length</code>	Calculates the length of the vector.
936	<code>squared_length</code>	Calculates the squared length of the vector.

26.20.3 Tvector4_extended.init_zero

Synopsis: Initializes the vector and sets its elements to zero

Declaration: `constructor init_zero`

Visibility: default

26.20.4 Tvector4_extended.init_one

Synopsis: Initializes the vector and sets its elements to one

Declaration: `constructor init_one`

Visibility: default

26.20.5 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

26.20.6 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` ([936](#)) if you are able to, as it is faster.

26.20.7 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`.

26.21 Tvector4_single

26.21.1 Description

The `Tvector4_single` object provides a vector of four single precision scalars.

26.21.2 Method overview

Page	Property	Description
936	<code>init</code>	Initializes the vector, setting its elements to the values passed to the constructor.
936	<code>init_one</code>	Initializes the vector and sets its elements to one
936	<code>init_zero</code>	Initializes the vector and sets its elements to zero
937	<code>length</code>	Calculates the length of the vector.
937	<code>squared_length</code>	Calculates the squared length of the vector.

26.21.3 Tvector4_single.init_zero

Synopsis: Initializes the vector and sets its elements to zero

Declaration: `constructor init_zero`

Visibility: default

26.21.4 Tvector4_single.init_one

Synopsis: Initializes the vector and sets its elements to one

Declaration: `constructor init_one`

Visibility: default

26.21.5 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

26.21.6 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` ([937](#)) if you are able to, as it is faster.

26.21.7 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 27

Reference for unit 'mmx'

27.1 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.

27.2 Constants, types and variables

27.2.1 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.

27.2.2 Types

`pmmxbyte = ^tmmxbyte`

Pointer to `tmmxbyte` (939) array type

`pmmxcardinal = ^tmmxcardinal`

Pointer to `tmmxcardinal` (939) array type

`pmmxinteger = ^tmmxinteger`

Pointer to `tmmxinteger` (939) array type

`pmmxlongint = ^tmmxlongint`

Pointer to `tmmxlongint` (939) array type

`pmmxshortint = ^tmmxshortint`

Pointer to `tmmxshortint` (939) array type

`pmmxsingle = ^tmmxsingle`

Pointer to `tmmxsingle` (939) array type

`pmmxword = ^tmmxword`

Pointer to `tmmxword` (939) 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

27.3 Procedures and functions

27.3.1 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
  dl : double;
  a : array[0..10000] of double;
  i : longint;
begin
  dl:=1.0;
  {$mmx+}
  { floating point data is used, but we do _no_ arithmetic }
  for i:=0 to 10000 do
    a[i]:=d2; { this is done with 64 bit moves }
  {$mmx-}
  emms; { clear fpu }
  { now we can do floating point arithmetic again }
end.
```

See also: `femms` ([940](#))

27.3.2 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` ([940](#))

Chapter 28

Reference for unit 'Mouse'

28.1 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.

28.2 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: `./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;
  WriteIn(MouseLog,'Start logging mouse events at: ',TimeStamp);
end;

Procedure StopMouseLogging;

begin
  WriteIn(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;
          WriteIn(MouseLog);
        end;
      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.

28.3 Constants, types and variables

28.3.1 Constants

`errMouseBase = 1030`

Base for mouse error codes.

`errMouseInitError = errMouseBase + 0`

Mouse initialization error

`errMouseNotImplemented = errMouseBase + 1`

Mouse driver not implemented.

MouseDown = \$0001

Mouse button down event signal.

MouseMove = \$0004

Mouse cursor move event signal.

MouseUp = \$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.

28.3.2 Types

PMouseEvent = ^TMouseEvent

Pointer to TMouseEvent ([945](#)) record.

```
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 `TMouseDown` record is used to implement a mouse driver in the `SetMouseDown` (950) function. Its fields must be filled in before calling the `SetMouseDown` (950) function.

```

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.

28.3.3 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.

28.4 Procedures and functions

28.4.1 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 ([949](#)), DoneMouse ([946](#))

Listing: ./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.

28.4.2 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 ([945](#)), InitMouse ([949](#))

28.4.3 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` (947), `GetMouseX` (948), `GetMouseY` (948)

Listing: `./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.

28.4.4 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` (941).

For an example, see the section on writing a custom mouse driver, `mousedrv` (941)

Errors: None.

See also: `SetMouseDriver` (950)

28.4.5 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` (950).

Errors: None.

See also: [GetMouseButtons \(946\)](#), [GetMouseX \(948\)](#), [GetMouseY \(948\)](#)

28.4.6 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 \(946\)](#), [GetMouseEvent \(947\)](#), [GetMouseY \(948\)](#)

Listing: `./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.

28.4.7 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 \(948\)](#)

Errors: None.

See also: [GetMouseButtons \(946\)](#), [GetMouseEvent \(947\)](#), [GetMouseX \(948\)](#)

28.4.8 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` ([951](#))

Listing: `./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.
```

28.4.9 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` (946) 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` (946), `DetectMouse` (945)

28.4.10 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` (947), `PutMouseEvent` (950)

28.4.11 PutMouseEvent

Synopsis: Put a mouse event in the venet queue.

Declaration: `procedure PutMouseEvent (const MouseEvent: TMouseEvent)`

Visibility: default

Description: `PutMouseEvent` adds `MouseEvent` to the input queue. The next call to `GetMouseEvent` (947) 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` (947), `PollMouseEvent` (950)

28.4.12 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` (949) 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` (941).

For an example, see `mousedrv` (941)

See also: [InitMouse \(949\)](#), [DoneMouse \(946\)](#), [GetMouseDriver \(947\)](#)

28.4.13 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 \(948\)](#), [GetMouseY \(948\)](#)

Listing: `./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.

28.4.14 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 \(949\)](#)

Errors: None.

See also: [HideMouse \(949\)](#)

Chapter 29

Reference for unit 'Objects'

29.1 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.

29.2 Constants, types and variables

29.2.1 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 (Pointer)`

Maximum collection size (in items)

`MaxPtrs = MaxBytes div (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 (Word)

Maximum data size (in words)

RCollection : TStreamRec = (ObjType: 50; VmtLink: (^TCollection)); Load: @TCollection

Default stream record for the TCollection (969) object.

RStrCollection : TStreamRec = (ObjType: 69; VmtLink: (^TStrCollection)); Load: @TStrCollection

Default stream record for the TStrCollection (1008) object.

RStringCollection : TStreamRec = (ObjType: 51; VmtLink: (^TStringCollection)); Load: @TStringCollection

Default stream record for the TStringCollection (1018) object.

RStringList : TStreamRec = (ObjType: 52; VmtLink: (^TStringList)); Load: @TStringList

Default stream record for the TStringList (1020) object.

RStrListMaker : TStreamRec = (ObjType: 52; VmtLink: (^TStrListMaker)); Load: Nil; S

Default stream record for the TStrListMaker (1022) 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).

29.2.2 Types

`AsciiZ = Array[0..255] of Char`

Filename - null terminated array of characters.

`FNameStr = String`

Filename - shortstring version.

```
LongRec = packed record
  Hi : Word;
  Lo : Word;
end
```

Record describing a longint (in Words)

```
PBufStream = ^TBufStream
```

Pointer to TBufStream (965) object.

```
PByteArray = ^TByteArray
```

Pointer to TByteArray (957)

```
PCharSet = ^TCharSet
```

Pointer to TCharSet (957).

```
PCollection = ^TCollection
```

Pointer to TCollection (969) object.

```
PDosStream = ^TDosStream
```

Pointer to TDosStream (983) object.

```
PItemList = ^TItemList
```

Pointer to TItemList (957) object.

```
PMemoryStream = ^TMemoryStream
```

Pointer to TMemoryStream (988) object.

```
PObject = ^TObject
```

Pointer to TObject (990) object.

```
PPoint = ^TPoint
```

Pointer to TPoint (992) record.

```
PPointerArray = ^TPointerArray
```

Pointer to TPointerArray (957)

```
PRect = ^TRect
```

Pointer to TRect (992) object.

`PResourceCollection = ^TResourceCollection`

Pointer to `TResourceCollection` (998) object.

`PResourceFile = ^TResourceFile`

Pointer to `TResourceFile` (999) object.

`PSortedCollection = ^TSortedCollection`

Pointer to `TSortedCollection` (1002) object.

`PStrCollection = ^TStrCollection`

Pointer to `TStrCollection` (1008) object.

`PStream = ^TStream`

Pointer type to `TStream` (1010)

`PStreamRec = ^TStreamRec`

Pointer to `TStreamRec` (957)

`PStrIndex = ^TStrIndex`

Pointer to `TStrIndex` (957) array.

`PString = PShortString`

Pointer to a shortstring.

`PStringCollection = ^TStringCollection`

Pointer to `TStringCollection` (1018) object.

`PStringList = ^TStringList`

Pointer to `TStringList` (1020) object.

`PStrListMaker = ^TStrListMaker`

Pointer to `TStrListMaker` (1022) object.

```
PTrRec = packed record
  Ofs : Word;
  Seg : Word;
end
```

Record describing a pointer to a memory location.

PUnSortedStrCollection = ^TUnSortedStrCollection

Pointer to TUnSortedStrCollection ([1023](#)) object.

PWordArray = ^TWordArray

Pointer to TWordArray ([958](#))

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 ([969](#))

TPointerArray = Array[0..MaxPtrs-1] of Pointer

Array with maximum allowed number of pointers

```
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.

TStrIndex = Array[0..9999] of TStrIndexRec

Pointer array type used in a TStringList ([1020](#))

```
TStrIndexRec = packed record
  Key : Sw_Word;
  Count : Word;
  Offset : Word;
end
```

Record type used in a TStringList (1020) to store the strings

```
TWordArray = Array[0..MaxWords-1] of Word
```

Array with maximum allowed number of words.

```
WordRec = packed record
  Hi : Byte;
  Lo : Byte;
end
```

Record describing a Word (in bytes)

29.2.3 Variables

```
invalidhandle : THandle
```

Value for invalid handle. Initial value for file stream handles or when the stream is closed.

29.3 Procedures and functions

29.3.1 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.

29.3.2 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` (960), `CallPointerMethod` (959), `CallVoidLocal` (960), `CallPointerLocal` (959), `CallVoidMethodLocal` (961), `CallPointerMethodLocal` (959)

29.3.3 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` (959), `CallVoidMethod` (960), `CallVoidLocal` (960), `CallVoidMethodLocal` (961), `CallPointerMethodLocal` (959), `CallVoidConstructor` (960), `CallPointerConstructor` (958)

29.3.4 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` (960), `CallVoidLocal` (960), `CallPointerLocal` (959), `CallVoidMethodLocal` (961), `CallPointerMethodLocal` (959), `CallVoidConstructor` (960), `CallPointerConstructor` (958)

29.3.5 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 \(959\)](#), [CallVoidMethod \(960\)](#), [CallPointerLocal \(959\)](#), [CallVoidLocal \(960\)](#), [CallVoidMethodLocal \(961\)](#), [CallVoidConstructor \(960\)](#), [CallPointerConstructor \(958\)](#)

29.3.6 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 \(958\)](#), [CallPointerMethod \(959\)](#), [CallVoidLocal \(960\)](#), [CallPointerLocal \(959\)](#), [CallVoidMethodLocal \(961\)](#), [CallPointerMethodLocal \(959\)](#)

29.3.7 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 \(959\)](#), [CallVoidMethod \(960\)](#), [CallPointerLocal \(959\)](#), [CallVoidMethodLocal \(961\)](#), [CallPointerMethodLocal \(959\)](#), [CallVoidConstructor \(960\)](#), [CallPointerConstructor \(958\)](#)

29.3.8 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 \(959\)](#), [CallVoidLocal \(960\)](#), [CallPointerLocal \(959\)](#), [CallVoidMethodLocal \(961\)](#), [CallPointerMethodLocal \(959\)](#), [CallVoidConstructor \(960\)](#), [CallPointerConstructor \(958\)](#)

29.3.9 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` (959), `CallVoidMethod` (960), `CallPointerLocal` (959), `CallVoidLocal` (960), `CallPointerMethodLocal` (959), `CallVoidConstructor` (960), `CallPointerConstructor` (958)

29.3.10 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` (962).

Errors: None.

See also: `NewStr` (962), `SetStr` (964)

29.3.11 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` (961)

29.3.12 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` (961)

29.3.13 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` ([961](#)), `SetStr` ([964](#))

Listing: `./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.

```

29.3.14 RegisterObjects

Synopsis: Register standard objects.

Declaration: `procedure RegisterObjects`

Visibility: default

Description: `RegisterObjects` registers the following objects for streaming:

1. `TCollection`, see `TCollection` ([969](#)).
2. `TStringCollection`, see `TStringCollection` ([1018](#)).
3. `TStrCollection`, see `TStrCollection` ([1008](#)).

Errors: None.

See also: `RegisterType` ([963](#))

29.3.15 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: `./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.

```

29.3.16 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` (962), 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` ([961](#)), `NewStr` ([962](#))

29.4 TBufStream

29.4.1 Description

`TBufStream` 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.

29.4.2 Method overview

Page	Property	Description
966	<code>Close</code>	Flush data and Close the file.
966	<code>Done</code>	Close the file and cleans up the instance.
966	<code>Flush</code>	FLush data from buffer, and write it to stream.
965	<code>Init</code>	Initialize an instance of <code>TBufStream</code> and open the file.
968	<code>Open</code>	Open the file if it is closed.
968	<code>Read</code>	Read data from the file to a buffer in memory.
967	<code>Seek</code>	Set current position in file.
967	<code>Truncate</code>	Flush buffer, and truncate the file at current position.
968	<code>Write</code>	Write data to the file from a buffer in memory.

29.4.3 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.

stOpenRead 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` ([966](#)).

Errors: On error, `Status` is set to `stInitError`, and `ErrorInfo` is set to the dos error code.

See also: `TDosStream.Init` ([984](#)), `TBufStream.Done` ([966](#))

29.4.4 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` (966).

Errors: None.

See also: `TDosStream.Done` (984), `TBufStream.Init` (965), `TBufStream.Close` (966)

29.4.5 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` (966) it does not clean up the instance of `TBufStream`

For an example see `TBufStream.Flush` (966).

Errors: None.

See also: `TStream.Close` (1014), `TBufStream.Init` (965), `TBufStream.Done` (966)

29.4.6 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` (968) for more info on the errors.

See also: `TStream.Close` (1014), `TBufStream.Init` (965), `TBufStream.Done` (966)

Listing: `./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.

```

29.4.7 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` (985).

Errors: Errors can be those of `Flush` (966) or `TDosStream.Truncate` (985).

See also: `TStream.Truncate` (1015), `TDosStream.Truncate` (985), `TStream.GetSize` (1012)

29.4.8 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` (1016);

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` (1016), `TStream.GetPos` (1012)

29.4.9 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` (966) call.

For an example, see `TDosStream.Open` (987).

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` (1014), `TBufStream.Close` (966)

29.4.10 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` (1017).

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` (1017), `TBufStream.Write` (968)

29.4.11 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` (1017).

Errors: In case of an error, `Status` is set to `StWriteError`, and `ErrorInfo` gets the OS specific error.

See also: `TStream.Write` (1017), `TBufStream.Read` (968)

29.5 TCollection

29.5.1 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.

29.5.2 Method overview

Page	Property	Description
971	<code>At</code>	Return the item at a certain index.
979	<code>AtDelete</code>	Delete item at certain position.
978	<code>AtFree</code>	Free an item at the indicates position, calling it's destructor.
982	<code>AtInsert</code>	Insert an element at a certain position in the collection.
982	<code>AtPut</code>	Set collection item, overwriting an existing value.
978	<code>Delete</code>	Delete an item from the collection, but does not destroy it.
976	<code>DeleteAll</code>	Delete all elements from the collection. Objects are not destroyed.
970	<code>Done</code>	Clean up collection, release all memory.
981	<code>Error</code>	Set error code.
973	<code>FirstThat</code>	Return first item which matches a test.
980	<code>ForEach</code>	Execute procedure for each item in the list.
977	<code>Free</code>	Free item from collection, calling it's destructor.
975	<code>FreeAll</code>	Release all objects from the collection.
979	<code>FreeItem</code>	Destroy a non-nil item.
972	<code>GetItem</code>	Read one item off the stream.
971	<code>IndexOf</code>	Find the position of a certain item.
969	<code>Init</code>	Instantiate a new collection.
977	<code>Insert</code>	Insert a new item in the collection at the end.
973	<code>LastThat</code>	Return last item which matches a test.
970	<code>Load</code>	Initialize a new collection and load collection from a stream.
974	<code>Pack</code>	Remove all <code>>Nil</code> pointers from the collection.
983	<code>PutItem</code>	Put one item on the stream
981	<code>SetLimit</code>	Set maximum number of elements in the collection.
983	<code>Store</code>	Write collection to a stream.

29.5.3 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` ([980](#)).

Errors: None.

See also: `TCollection.Load` ([970](#)), `TCollection.Done` ([970](#))

29.5.4 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 (972).

See also: TCollection.Init (969), TCollection.GetItem (972), TCollection.Done (970)

Listing: ./objectex/ex22.pp

Program ex22;

{ Program to demonstrate the TCollection.Load method }

Uses Objects, MyObject; *{ For TMyObject definition and registration }*

Var C : PCollection;
 M : PMyObject;
 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.

29.5.5 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 (980).

Errors: None.

See also: `TCollection.Init` (969), `TCollection.FreeAll` (975)

29.5.6 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, `seep1{Error}{TCollection.Error}` is called with `coIndexError` and `Index` as arguments, resulting in a run-time error.

See also: `TCollection.Insert` (977)

Listing: `./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.

29.5.7 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 ([971](#)), TCollection.GetItem ([972](#)), TCollection.Insert ([977](#))

Listing: ./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.

29.5.8 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 ([1010](#)).

See also: TStream.Get ([1010](#)), TCollection.Store ([983](#))

29.5.9 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` ([973](#))

Listing: `./objectex/ex25.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`;

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 ('Last one for which Field < 56 has index (should be 54) : ',
 `C^.IndexOf (C^.LastThat (@CheckField))`);
 `C^.FreeAll`;
 Dispose (`C`, `Done`);
end.

29.5.10 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` ([973](#))

Listing: `./objectex/ex26.pp`

Program `ex21`;

{ Program to demonstrate the TCollection.FirstThat 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 ('first one for which Field > 56 has index (should be 56) : ',
 `C^.IndexOf(C^.FirstThat(@CheckField))`);
 `C^.FreeAll`;
 Dispose(`C`, `Done`);
end.

29.5.11 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` ([981](#))

Listing: ./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.

29.5.12 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` (976), `TCollection.FreeItem` (979)

Listing: ./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;
  Writeln('Added 100 Items. ');
  C^.FreeAll;
  Writeln('Freed all objects. ');
  Dispose(C, Done);
end.

```

29.5.13 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` (975), `DeleteAll` doesn't call the destructor of the objects.

Errors: None.

See also: `TCollection.FreeAll` (975), `TCollection.Delete` (978)

Listing: `./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.

```

29.5.14 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` ([979](#))

Listing: `./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.

```

29.5.15 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` (982), `TCollection.AtPut` (982)

29.5.16 `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` (977) call is provided.

Errors: If the `Item` is not in the collection, `Error` will be called with `coIndexError`.

See also: `TCollection.AtDelete` (979), `TCollection.Free` (977)

Listing: `./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.

29.5.17 `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 (981) is called with `CoIndexError`.

See also: `TCollection.Free` (977), `TCollection.AtDelete` (979)

Listing: ./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.

```

29.5.18 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` (977), `TCollection.AtFree` (978)

29.5.19 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 (981) is called with `CoIndexError`.

See also: `TCollection.Delete` (978)

Listing: ./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.

29.5.20 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` (973), `TCollection.LastThat` (973)

Listing: ./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.

```

29.5.21 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 (974).

Errors: None.

See also: `TCollection.Init` (969)

29.5.22 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` (958)

29.5.23 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` (974).

Errors: If `Index` isn't valid then `Error` (981) is called with `CoIndexError`.

29.5.24 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` (981) is called with `CoIndexError`. If the collection fails to expand, then `coOverflow` is passed to `Error`.

See also: `TCollection.Insert` (977)

Listing: `./objectex/ex34.pp`

Program `ex34`;

{ Program to demonstrate the TCollection.AtInsert 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
 `Randomize`;
 `C`:=`New`(`PCollection`, `Init`(120,10));
 `WriteLn` ('Inserting 100 records at random places.');
 For `I`:=1 **to** 100 **do**
 begin
 `M`:=`New`(`PMyObject`, `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.

```

29.5.25 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 writeing 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` (970).

For an example, see `TCollection.Load` (970).

Errors: Errors returned are those by `TStream.Put` (1015).

See also: `TCollection.Load` (970), `TCollection.PutItem` (983)

29.5.26 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 internaly by the `TCollection` object, and should not be called directly.

Errors: Errors are those returned by `TStream.Put` (1015).

See also: `Store` (983), `GetItem` (972)

29.6 TDosStream

29.6.1 Description

`TDosStream` is a stream that stores it's contents in a file. it overrides a couple of methods of `TStream` (1010) for this.

In addition to the fields inherited from `TStream` (see `TStream` (1010)), 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` (965).

29.6.2 Method overview

Page	Property	Description
985	Close	Close the file.
984	Done	Closes the file and cleans up the instance.
984	Init	Instantiate a new instance of TDosStream.
987	Open	Open the file stream
987	Read	Read data from the stream to a buffer.
986	Seek	Set file position.
985	Truncate	Truncate the file on the current position.
988	Write	Write data from a buffer to the stream.

29.6.3 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.

stOpenRead and write access.

For an example, see `TDosStream.Truncate` ([985](#)).

Errors: On error, `Status` (??) is set to `stInitError`, and `ErrorInfo` is set to the dos error code.

See also: `TDosStream.Done` ([984](#))

29.6.4 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` ([985](#)).

Errors: None.

See also: `TDosStream.Init` ([984](#)), `TDosStream.Close` ([985](#))

29.6.5 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` (984) it does not clean up the instance of `TDosStream`

For an example, see `TDosStream.Open` (987).

Errors: None.

See also: `TStream.Close` (1014), `TDosStream.Init` (984), `TDosStream.Done` (984)

29.6.6 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` (1015), `TStream.GetSize` (1012)

Listing: `./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.

```

29.6.7 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` ([1016](#)), `TStream.GetPos` ([1012](#))

Listing: `./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.

```

29.6.8 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` (985) 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` (1014), `TDosStream.Close` (985)

Listing: `./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.

29.6.9 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` (1017).

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` (1017), `TDosStream.Write` (988)

29.6.10 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` ([1017](#)).

Errors: In case of an error, `Status` is set to `StWriteError`, and `ErrorInfo` gets the OS specific error.

See also: `TStream.Write` ([1017](#)), `TDosStream.Read` ([987](#))

29.7 TMemoryStream

29.7.1 Description

The `TMemoryStream` object implements a stream that stores its 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` ([1010](#))

29.7.2 Method overview

Page	Property	Description
989	<code>Done</code>	Clean up memory and destroy the object instance.
988	<code>Init</code>	Initialize memory stream, reserves memory for stream data.
990	<code>Read</code>	Read data from the stream to a location in memory.
989	<code>Truncate</code>	Set the stream size to the current position.
990	<code>Write</code>	Write data to the stream.

29.7.3 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` ([1018](#)).

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` ([989](#))

29.7.4 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` ([1018](#)).

Errors: `None`.

See also: `TMemoryStream.Init` ([988](#))

29.7.5 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` ([1015](#))

Listing: `./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.

29.7.6 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` ([1017](#)).

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` ([1017](#)), `TMemoryStream.Write` ([990](#))

29.7.7 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` ([988](#))) as needed.

For an example, see `TStream.Read` ([1017](#)).

Errors: If the stream cannot allocate more memory, then the status is set to `stWriteError`

See also: `TStream.Write` ([1017](#)), `TMemoryStream.Read` ([990](#))

29.8 TObject

29.8.1 Description

This type serves as the basic object for all other objects in the `Objects` unit.

29.8.2 Method overview

Page	Property	Description
992	<code>Done</code>	Destroy an object.
991	<code>Free</code>	Destroy an object and release all memory.
990	<code>Init</code>	Construct (initialize) a new object
991	<code>Is_Object</code>	Check whether a pointer points to an object.

29.8.3 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 \(991\)](#)

Errors: None.

See also: [TObject.Free \(991\)](#), [TObject.Done \(992\)](#)

29.8.4 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 \(990\)](#), [TObject.Done \(992\)](#)

Listing: `./objectex/ex7.pp`

```

program ex7;

  { Program to demonstrate the TObject.Free call }

Uses Objects;

Var O : TObject;

begin
  // Allocate memory for object.
  O:=New(TObject, Init);
  // Free memory of object.
  O^.free;
end.
```

29.8.5 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.

29.8.6 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` (991) method.

The destructore `Done` does not free the memory occupied by the object.

Errors: `None`.

See also: `TObject.Free` (991), `TObject.Init` (990)

Listing: `./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.

```

29.9 TPoint

29.9.1 Description

Record describing a point in a 2 dimensional plane.

29.10 TRect

29.10.1 Description

Describes a rectangular region in a plane.

29.10.2 Method overview

Page	Property	Description
997	Assign	Set rectangle corners.
994	Contains	Determine if a point is inside the rectangle
994	Copy	Copy cornerpoints from another rectangle.
993	Empty	Is the surface of the rectangle zero
994	Equals	Do the corners of the rectangles match
997	Grow	Expand rectangle with certain size.
995	Intersect	Reduce rectangle to intersection with another rectangle
996	Move	Move rectangle along a vector.
995	Union	Enlarges rectangle to encompass another rectangle.

29.10.3 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` ([994](#)), `TRect.Contains` ([994](#))

Listing: `./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.

29.10.4 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` (993)

Errors: None.

See also: `TRect.Empty` (993), `TRect.Contains` (994)

29.10.5 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` (995), `TRect.Equals` (994)

29.10.6 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` (997)

Listing: `./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.

```

29.10.7 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` ([995](#))

Listing: `./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.

```

29.10.8 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 \(995\)](#)

Listing: ./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.
```

29.10.9 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 \(997\)](#)

Listing: ./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.

```

29.10.10 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` ([996](#))

Listing: `./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.

```

29.10.11 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` (994).

Errors: None.

See also: `TRect.Copy` (994)

29.11 TResourceCollection

29.11.1 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.

29.11.2 Method overview

Page	Property	Description
999	<code>FreeItem</code>	Release memory occupied by item.
999	<code>GetItem</code>	Read an item from the stream.
998	<code>KeyOf</code>	Return the key of an item in the collection.
999	<code>PutItem</code>	Write an item to the stream.

29.11.3 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` (1019)

29.11.4 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 (970)` method.

Errors: Errors returned are those by `TStream.Read (1017)`

See also: `TCollection.Load (970)`, `TStream.Read (1017)`

29.11.5 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 (979)`

29.11.6 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 (983)` method.

Errors: Errors returned are those by `TStream.Write (1017)`.

See also: `TCollection.Store (983)`

29.12 TResourceFile

29.12.1 Description

`TResourceFile (999)` represents the resources in a binary file image.

29.12.2 Method overview

Page	Property	Description
1000	Count	Number of resources in the file
1002	Delete	Delete a resource from the file
1000	Done	Destroy the instance and remove it from memory.
1001	Flush	Writes the resources to the stream.
1001	Get	Return a resource by key name.
1000	Init	Instantiate a new instance.
1001	KeyAt	Return the key of the item at a certain position.
1002	Put	Set a resource by key name.
1001	SwitchTo	Write resources to a new stream.

29.12.3 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` ([1000](#))

29.12.4 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` ([1000](#))

29.12.5 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` ([1000](#))

29.12.6 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` ([1001](#))

29.12.7 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`

29.12.8 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` ([1017](#)) and `TStream.Write` ([1017](#)).

See also: `TResourceFile.Flush` ([1001](#))

29.12.9 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` ([1016](#)) and `TStream.Write` ([1017](#)).

See also: `TResourceFile.SwitchTo` ([1001](#))

29.12.10 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` ([1001](#))

29.12.11 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` ([1015](#)) and `TStream.Seek`

See also: `Get` ([1001](#))

29.13 TSortedCollection

29.13.1 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` ([1004](#)) method.

Because the collection is ordered, `TSortedCollection` overrides some `TCollection` methods, to provide faster routines for lookup.

The `Compare` ([1004](#)) 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.

29.13.2 Method overview

Page	Property	Description
1004	Compare	Compare two items in the collection.
1004	IndexOf	Return index of an item in the collection.
1003	Init	Instantiates a new instance of a <code>TSortedCollection</code>
1006	Insert	Insert new item in collection.
1003	KeyOf	Return the key of an item
1003	Load	Instantiates a new instance of a <code>TSortedCollection</code> and loads it from stream.
1005	Search	Search for item with given key.
1007	Store	Write the collection to the stream.

29.13.3 `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` ([969](#))) 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` ([1003](#)), `TCollection.Done` ([970](#))

29.13.4 `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` ([970](#))) 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` ([970](#)).

Errors: None.

See also: `TSortedCollection.Init` ([1003](#)), `TCollection.Done` ([970](#))

29.13.5 `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` ([1004](#)), `TSortedCollection.Compare` ([1004](#))

29.13.6 `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` ([971](#))

Errors: None.

See also: `TSortedCollection.Search` ([1005](#)), `TSortedCollection.Compare` ([1004](#))

29.13.7 `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` ([1005](#)) method and in the `Insert` ([1006](#)) 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` ([1004](#)), `TSortedCollection.Search` ([1005](#))

Listing: `./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.

29.13.8 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 (1004) function to implement this search.

If the item is found, Search returns True, otherwise False is returned.

Errors: None.

See also: TCollection.IndexOf (971)

Listing: ./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.

```

29.13.9 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` (982), 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` (982)

Listing: `./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.

```

29.13.10 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` (983), and then writing the `Duplicates` flag to the stream.

After a `Store`, the collection can be loaded from the stream with the constructor `Load` (1003)

For an example, see `TCollection.Load` (970).

Errors: Errors can be those of `TStream.Put` (1015).

See also: `TSortedCollection.Load` (1003)

29.14 TStrCollection

29.14.1 Description

The `TStrCollection` object manages a sorted collection of null-terminated strings (pchar strings). To this end, it overrides the `Compare` (1004) method of `TSortedCollection`, and it introduces methods to read/write strings from a stream.

29.14.2 Method overview

Page	Property	Description
1008	<code>Compare</code>	Compare two strings in the collection.
1009	<code>FreeItem</code>	Free null-terminated string from the collection.
1009	<code>GetItem</code>	Read a null-terminated string from the stream.
1009	<code>PutItem</code>	Write a null-terminated string to the stream.

29.14.3 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` (1004)

Listing: `./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.

```

29.14.4 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` ([1011](#)).

See also: `TStrCollection.PutItem` ([1009](#))

29.14.5 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` ([979](#))

29.14.6 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` ([1016](#)).

See also: `TStrCollection.GetItem` ([1009](#))

29.15 TStream

29.15.1 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` ([956](#)), `TDosStream` ([983](#)), `TMemoryStream` ([988](#))

29.15.2 Method overview

Page	Property	Description
1014	<code>Close</code>	Close the stream
1018	<code>CopyFrom</code>	Copy data from another stream.
1016	<code>Error</code>	Set stream status
1015	<code>Flush</code>	Flush the stream data from the buffer, if any.
1010	<code>Get</code>	Read an object definition from the stream.
1012	<code>GetPos</code>	Return current position in the stream
1012	<code>GetSize</code>	Return the size of the stream.
1010	<code>Init</code>	Constructor for <code>TStream</code> instance
1014	<code>Open</code>	Open the stream
1015	<code>Put</code>	Write an object to the stream.
1017	<code>Read</code>	Read data from stream to buffer.
1013	<code>ReadStr</code>	Read a shortstring from the stream.
1014	<code>Reset</code>	Reset the stream
1016	<code>Seek</code>	Set stream position.
1011	<code>StrRead</code>	Read a null-terminated string from the stream.
1016	<code>StrWrite</code>	Write a null-terminated string to the stream.
1015	<code>Truncate</code>	Truncate the stream size on current position.
1017	<code>Write</code>	Write a number of bytes to the stream.
1016	<code>WriteStr</code>	Write a pascal string to the stream.

29.15.3 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`.

29.15.4 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` ([1015](#))

Listing: `./objectex/ex9.pp`

Program `ex9`;

{ Program to demonstrate TStream.Get and TStream.Put }

Uses `Objects, MyObject`; *{ Definition and registration of TMyObject }*

Var `Obj : PMyObject`;
 `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:=PMyObject(S^.Get);
WriteLn ('Field Value : ', Obj^.GetField);
Dispose(Obj, Done);

```

end.

29.15.5 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` ([1016](#)), `TStream.ReadStr` ([1013](#))

Listing: `./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.

```

29.15.6 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` ([1016](#)), `TStream.GetSize` ([1012](#))

Listing: `./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.

```

29.15.7 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` ([1016](#)), `TStream.GetPos` ([1012](#))

Listing: `./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.
```

29.15.8 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` ([1011](#))

Listing: `./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.

```

29.15.9 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` ([987](#)).

Errors: None.

See also: `TStream.Close` ([1014](#)), `TStream.Reset` ([1014](#))

29.15.10 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` ([987](#)).

Errors: None.

See also: `TStream.Open` ([1014](#)), `TStream.Reset` ([1014](#))

29.15.11 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` ([1014](#)), `TStream.Close` ([1014](#))

29.15.12 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` (966).

Errors: None.

See also: `TStream.Truncate` (1015)

29.15.13 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` (985).

Errors: None.

See also: `TStream.Seek` (1016)

29.15.14 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` (963).

After the object has been written, it can be read again with `Get` (1010).

For an example, see `TStream.Get` (1010);

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` (1010)

29.15.15 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 65355 bytes long.

For an example, see `TStream.StrRead` (1011).

Errors: None.

See also: `TStream.WriteString` (1016), `TStream.StrRead` (1011), `TStream.ReadStr` (1013)

29.15.16 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` (1013).

Errors: None.

See also: `TStream.StrWrite` (1016), `TStream.StrRead` (1011), `TStream.ReadStr` (1013)

29.15.17 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` (986).

Errors: If `Pos` is larger than the stream size, `Status` is set to `StSeekError`.

See also: `TStream.GetPos` (1012), `TStream.GetSize` (1012)

29.15.18 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.

29.15.19 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 (1017), TStream.ReadStr (1013), TStream.StrRead (1011)

Listing: ./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.

```

29.15.20 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 (1017).

Errors: No checking is done to see if Buf actually contains Count bytes.

See also: TStream.Read (1017), TStream.WriteString (1016), TStream.StrWrite (1016)

29.15.21 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` (1017) method to read the data, and the `Write` (1017) method to write in the current stream.

Errors: None.

See also: `Read` (1017), `Write` (1017)

Listing: `./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.

29.16 TStringCollection

29.16.1 Description

The `TStringCollection` object manages a sorted collection of pascal strings. To this end, it overrides the `Compare` (1004) method of `TSortedCollection`, and it introduces methods to read/write strings from a stream.

29.16.2 Method overview

Page	Property	Description
1019	Compare	Compare two strings in the collection.
1020	FreeItem	Dispose a string in the collection from memory.
1019	GetItem	Get string from the stream.
1020	PutItem	Write a string to the stream.

29.16.3 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` ([1013](#)).

See also: `TStringCollection.PutItem` ([1020](#))

29.16.4 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` ([1004](#))

Listing: `./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.

```

29.16.5 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` ([979](#))

29.16.6 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` ([1016](#)).

See also: `TStringCollection.GetItem` ([1019](#))

29.17 TStringList

29.17.1 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` ([963](#)) function, you cannot register the `TStrListMaker` object.

29.17.2 Method overview

Page	Property	Description
1021	Done	Clean up the instance
1021	Get	Return a string by key name
1021	Load	Load stringlist from stream.

29.17.3 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` ([1021](#))

29.17.4 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` ([1021](#)), `TObject.Done` ([992](#))

29.17.5 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` ([1022](#))

29.18 TStrListMaker

29.18.1 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` (963) function, you cannot register the `TStringList` object.

29.18.2 Method overview

Page	Property	Description
1022	<code>Done</code>	Clean up the instance and free all related memory.
1022	<code>Init</code>	Instantiate a new instance of <code>TStrListMaker</code>
1022	<code>Put</code>	Add a new string to the list with associated key.
1023	<code>Store</code>	Write the strings to the stream.

29.18.3 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` (990), `TStrListMaker.Done` (1022)

29.18.4 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` (992), `TStrListMaker.Init` (1022)

29.18.5 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` ([1023](#)) method.

Errors: None.

See also: `TStrListMaker.Store` ([1023](#))

29.18.6 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` ([1021](#)), `TStrListMaker.Put` ([1022](#))

29.19 TUnSortedStrCollection

29.19.1 Description

The `TUnSortedStrCollection` object manages an unsorted list of strings. To this end, it overrides the `TSortedCollection.Insert` ([1006](#)) method to add strings at the end of the collection, rather than in the alphabetically correct position.

Take care, the `Search` ([1005](#)) and `IndexOf` ([971](#)) methods will not work on an unsorted string collection.

29.19.2 Method overview

Page	Property	Description
1023	<code>Insert</code>	Insert a new string in the collection.

29.19.3 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` ([977](#))

Listing: `./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 30

Reference for unit 'objpas'

30.1 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.

30.2 Constants, types and variables

30.2.1 Constants

`MaxInt = MaxLongint`

Maximum value for Integer (1025) type.

30.2.2 Types

`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..$efffffff] of Integer`

Generic array of integer (1025)

`PInteger = ^Integer`

Pointer to Integer (1025) type.

`PIntegerArray = ^IntegerArray`

Pointer to TIntegerArray (1026) type.

`PointerArray = Array[0..512*1024*1024-2] of Pointer`

Generic Array of pointers.

`PPointerArray = ^PointerArray`

Pointer to PointerArray ([1026](#))

`PString = PAnsiString`

Pointer to ansistring type.

`TBoundArray = Array of Integer`

Array of integer, used in interfaces.

`TIntegerArray = IntegerArray`

Alias for IntegerArray ([1025](#))

`TPointerArray = PointerArray`

Alias for PointerArray ([1026](#))

Chapter 31

Reference for unit 'ports'

31.1 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. It's 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.

31.2 Constants, types and variables

31.2.1 Variables

`port : tport`

Default instance of type `TPort` ([1028](#)). 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` ([1028](#)). 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 ([1029](#)). 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 ([1029](#)). 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

31.3 tport

31.3.1 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 ([1027](#)), TPortW ([1029](#)), TPortL ([1029](#))

31.3.2 Property overview

Page	Property	Access	Description
1028	pp	rw	Access integer-sized port by port number

31.3.3 tport.pp

Synopsis: Access integer-sized port by port number

Declaration: Property pp[w: LongInt]: Byte; default

Visibility: public

Access: Read,Write

Description: Access integer-sized port by port number

31.4 tporth

31.4.1 Description

The `TPorth` 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` ([1028](#)), `TPort` ([1028](#)), `TPorth` ([1029](#))

31.4.2 Property overview

Page	Property	Access	Description
1029	<code>pp</code>	<code>rw</code>	Access Longint-sized port by port number

31.4.3 tporth.pp

Synopsis: Access Longint-sized port by port number

Declaration: `Property pp[w: LongInt]: LongInt; default`

Visibility: `public`

Access: `Read,Write`

Description: Access Longint-sized port by port number

31.5 portw

31.5.1 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` ([1028](#)), `TPort` ([1028](#)), `TPorth` ([1029](#))

31.5.2 Property overview

Page	Property	Access	Description
1029	<code>pp</code>	<code>rw</code>	Access word-sized port by port number

31.5.3 portw.pp

Synopsis: Access word-sized port by port number

Declaration: `Property pp[w: LongInt]: Word; default`

Visibility: `public`

Access: `Read,Write`

Description: Access word-sized port by port number

Chapter 32

Reference for unit 'printer'

32.1 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.

32.2 Constants, types and variables

32.2.1 Variables

`Lst` : `text`

`Lst` is the standard printing device.

On Linux, `Lst` is set up using `AssignLst ('/tmp/PID.lst')`.

32.3 Procedures and functions

32.3.1 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 \(1030\)](#)

Listing: ./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.

```

32.3.2 InitPrinter

Synopsis: Initialize the printer

Declaration: `procedure InitPrinter(const PrinterName: string)`

Visibility: default

Description: Initialize the printer

32.3.3 IsLstAvailable

Synopsis: Determine whether printer is available.

Declaration: `function IsLstAvailable : Boolean`

Visibility: default

Description: Determine whether printer is available.

Chapter 33

Reference for unit 'Sockets'

33.1 Used units

Table 33.1: Used units by unit 'Sockets'

Name	Page
BaseUnix	100
System	1118
unixtype	1623

33.2 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.

33.3 Constants, types and variables

33.3.1 Constants

`AF_APPLETALK = 5`

Address family Appletalk DDP

`AF_ASH = 18`

Address family: Ash

`AF_ATMPVC = 8`

Address family: ATM PVCs

`AF_ATMSVC = 20`

Address family: ATM SVCs

AF_AX25 = 3

Address family Amateur Radio AX.25

AF_BLUETOOTH = 31

Address family: Bluetooth sockets

AF_BRIDGE = 7

Address family Multiprotocol bridge

AF_DECnet = 12

Address family: Reserved for DECnet project.

AF_ECONET = 19

Address family: Acorn Econet

AF_INET = 2

Address family Internet IP Protocol

AF_INET6 = 10

Address family IP version 6

AF_IPX = 4

Address family Novell IPX

AF_IRDA = 23

Address family: IRDA sockets

AF_KEY = 15

Address family: PF_KEY key management API

AF_LLC = 26

Address family: Linux LLC

AF_LOCAL = 1

Address family: Unix socket

AF_MAX = 32

Address family Maximum value

AF_NETBEUI = 13

Address family: Reserved for 802.2LLC project

AF_NETLINK = 16

Address family: ?

AF_NETROM = 6

Address family Amateur radio NetROM

AF_PACKET = 17

Address family: Packet family

AF_PPPOX = 24

Address family: PPPoX sockets

AF_ROSE = 11

Address family: Amateur Radio X.25 PLP

AF_ROUTE = AF_NETLINK

Address family: Alias to emulate 4.4BSD.

AF_SECURITY = 14

Address family: Security callback pseudo AF

AF_SNA = 22

Address family: Linux SNA project

AF_TIPC = 30

Address family: TIPC sockets

AF_UNIX = 1

Address family Unix domain sockets

AF_UNSPEC = 0

Address family Not specified

AF_WANPIPE = 25

Address family: Wanpipe API Sockets

`AF_X25 = 9`

Address family Reserved for X.25 project

`EsockEACCESS = ESysEAcces`

Access forbidden error

`EsockEBADF = EsysEBADF`

Alias: bad file descriptor

`EsockEFAULT = EsysEFAULT`

Alias: an error occurred

`EsockEINTR = EsysEINTR`

Alias : operation interrupted

`EsockEINVAL = EsysEINVAL`

Alias: Invalid value specified

`EsockEMFILE = ESysEmfile`

Error code ?

`EsockEMSGSIZE = ESysEMsgSize`

Wrong message size error

`EsockENOBUFS = ESysENoBufs`

No buffer space available error

`EsockENOTCONN = ESysENotConn`

Not connected error

`EsockENOTSOCK = ESysENotSock`

File descriptor is not a socket error

`EsockEPROTONOSUPPORT = ESysEProtoNoSupport`

Protocol not supported error

`EsockEWOULDBLOCK = ESysEWouldBlock`

Operation would block error

INADDR_ANY = (0)

A bitmask matching any IP address on the local machine.

INADDR_NONE = (\$FFFFFFFF)

A bitmask matching no valid IP address

IPPROTO_AH = 51

authentication header.

IPPROTO_COMP = 108

Compression Header Protocol.

IPPROTO_DSTOPTS = 60

IPv6 destination options.

IPPROTO_EGP = 8

Exterior Gateway Protocol.

IPPROTO_ENCAP = 98

Encapsulation Header.

IPPROTO_ESP = 50

encapsulating security payload.

IPPROTO_FRAGMENT = 44

IPv6 fragmentation header.

IPPROTO_GRE = 47

General Routing Encapsulation.

IPPROTO_HOPOPTS = 0

IPv6 Hop-by-Hop options.

IPPROTO_ICMP = 1

Internet Control Message Protocol.

IPPROTO_ICMPV6 = 58

ICMPv6.

IPPROTO_IDP = 22

XNS IDP protocol.

IPPROTO_IGMP = 2

Internet Group Management Protocol.

IPPROTO_IP = 0

Dummy protocol for TCP.

IPPROTO_IPIP = 4

IPIP tunnels (older KA9Q tunnels use 94).

IPPROTO_IPV6 = 41

IPv6 header.

IPPROTO_MAX = 255

Maximum value for IPPROTO options

IPPROTO_MTP = 92

Multicast Transport Protocol.

IPPROTO_NONE = 59

IPv6 no next header.

IPPROTO_PIM = 103

Protocol Independent Multicast.

IPPROTO_PUP = 12

PUP protocol.

IPPROTO_RAW = 255

Raw IP packets.

IPPROTO_ROUTING = 43

IPv6 routing header.

IPPROTO_RSVP = 46

Reservation Protocol.

IPPROTO_SCTP = 132

Stream Control Transmission Protocol.

IPPROTO_TCP = 6

Transmission Control Protocol.

IPPROTO_TP = 29

SO Transport Protocol Class 4.

IPPROTO_UDP = 17

User Datagram Protocol.

IPV6_ADDRFORM = 1

Change the IPV6 address into a different address family. Deprecated

IPV6_ADD_MEMBERSHIP = IPV6_JOIN_GROUP

Undocumented Getsockopt option ?

IPV6_AUTHHDR = 10

GetSockOpt/SetSockopt: Deliver authentication header messages

IPV6_CHECKSUM = 7

Undocumented Getsockopt option ?

IPV6_DROP_MEMBERSHIP = IPV6_LEAVE_GROUP

Undocumented Getsockopt option ?

IPV6_DSTOPTS = 4

Deliver destination option control messages

IPV6_HOPLIMIT = 8

Deliver an integer containing the HOP count

IPV6_HOPOPTS = 3

Deliver hop option control messages

IPV6_IPSEC_POLICY = 34

Undocumented Getsockopt option ?

IPV6_JOIN_ANYCAST = 27

Undocumented Getsockopt option ?

IPV6_JOIN_GROUP = 20

GetSockOpt/SetSockopt: Control membership (join group) in multicast groups

IPV6_LEAVE_ANYCAST = 28

Undocumented Getsockopt option ?

IPV6_LEAVE_GROUP = 21

GetSockOpt/SetSockopt: Control membership (leave group) in multicast groups

IPV6_MTU = 24

GetSockOpt/SetSockopt: Get/Set the MTU for the socket

IPV6_MTU_DISCOVER = 23

GetSockOpt/SetSockopt: Get/Set Control path MTU Discovery on the socket

IPV6_MULTICAST_HOPS = 18

GetSockOpt/SetSockopt: Get/Set the multicast hop limit.

IPV6_MULTICAST_IF = 17

GetSockOpt/SetSockopt: Get/Set device for multicast packages on the socket.

IPV6_MULTICAST_LOOP = 19

GetSockOpt/SetSockopt: Control whether socket sees multicast packages that it has sent itself

IPV6_NEXTHOP = 9

sendmsg: set next hop for IPV6 datagram

IPV6_PKTINFO = 2

Change delivery options for incoming IPV6 datagrams

IPV6_PKTOPTIONS = 6

Undocumented Getsockopt option ?

IPV6_PMTUDISC_DO = 2

Always DF.

IPV6_PMTUDISC_DONT = 0

Never send DF frames.

IPV6_PMTUDISC_WANT = 1

Use per route hints.

IPV6_RECVERR = 25

GetSockOpt/SetSockopt: Control receiving of asynchronous error options

IPV6_ROUTER_ALERT = 22

GetSockOpt/SetSockopt: Get/Set Pass all forwarded packets containing router alert option

IPV6_RTHDR = 5

Deliver routing header control messages

IPV6_RTHDR_LOOSE = 0

Hop doesn't need to be neighbour.

IPV6_RTHDR_STRICT = 1

Hop must be a neighbour.

IPV6_RTHDR_TYPE_0 = 0

IPv6 Routing header type 0.

IPV6_RXDSTOPTS = IPV6_DSTOPTS

Undocumented Getsockopt option ?

IPV6_RXHOPOPTS = IPV6_HOPOPTS

Undocumented Getsockopt option ?

IPV6_RXSRCRT = IPV6_RTHDR

Undocumented Getsockopt option ?

IPV6_UNICAST_HOPS = 16

GetSockOpt/SetSockopt: Get/Set unicast hop limit

IPV6_V6ONLY = 26

GetSockOpt/SetSockopt: Handle IPV6 connections only

IPV6_XFRM_POLICY = 35

Undocumented Getssockopt option ?

IP_ADD_MEMBERSHIP = 35

add an IP group membership

IP_ADD_SOURCE_MEMBERSHIP = 39

join source group

IP_BLOCK_SOURCE = 38

block data from source

IP_DEFAULT_MULTICAST_LOOP = 1

Undocumented ?

IP_DEFAULT_MULTICAST_TTL = 1

Undocumented ?

IP_DROP_MEMBERSHIP = 36

drop an IP group membership

IP_DROP_SOURCE_MEMBERSHIP = 40

leave source group

IP_HDRINCL = 3

Header is included with data.

IP_MAX_MEMBERSHIPS = 20

Maximum group memberships for multicast messages

IP_MSFILTER = 41

Undocumented ?

IP_MTU_DISCOVER = 10

Undocumented ?

IP_MULTICAST_IF = 32

set/get IP multicast i/f

IP_MULTICAST_LOOP = 34

set/get IP multicast loopback

IP_MULTICAST_TTL = 33

set/get IP multicast ttl

IP_OPTIONS = 4

IP per-packet options.

IP_PKTINFO = 8

Undocumented ?

IP_PKTOPTIONS = 9

Undocumented ?

IP_PMTUDISC = 10

Undocumented ?

IP_PMTUDISC_DO = 2

Always DF.

IP_PMTUDISC_DONT = 0

Never send DF frames.

IP_PMTUDISC_WANT = 1

Use per route hints.

IP_RECVERR = 11

Undocumented ?

IP_RECVOPTS = 6

Receive all IP options w/datagram.

IP_RECVRETOPTS = IP_RETOPTS

Receive IP options for response.

IP_RECVTOS = 13

Undocumented ?

IP_RECVTTL = 12

Undocumented ?

IP_RETOPTS = 7

Set/get IP per-packet options.

IP_ROUTER_ALERT = 5

Undocumented ?

IP_TOS = 1

IP type of service and precedence.

IP_TTL = 2

IP time to live.

IP_UNBLOCK_SOURCE = 37

unblock data from source

MCAST_BLOCK_SOURCE = 43

block from given group

MCAST_EXCLUDE = 0

Undocumented ?

MCAST_INCLUDE = 1

Undocumented ?

MCAST_JOIN_GROUP = 42

join any-source group

MCAST_JOIN_SOURCE_GROUP = 46

join source-spec group

MCAST_LEAVE_GROUP = 45

leave any-source group

MCAST_LEAVE_SOURCE_GROUP = 47

leave source-spec group

MCAST_MSFILTER = 48

Undocumented ?

MCAST_UNBLOCK_SOURCE = 44

unblock from given group

MSG_CONFIRM = 0800

Send flags: Conform connection

MSG_CTRUNC = 0008

Receive flags: Control Data was discarded (buffer too small)

MSG_DONTROUTE = 0004

Send flags: don't use gateway

MSG_DONTWAIT = 0040

Receive flags: Non-blocking operation request.

MSG_EOF = MSG_FIN

Alias for MSG_FIN

MSG_EOR = 0080

Receive flags: End of record

MSG_ERRQUEERE = 2000

Receive flags: ?

MSG_FIN = 0200

Receive flags: ?

MSG_MORE = 8000

Receive flags: ?

MSG_NOSIGNAL = 4000

Receive flags: Suppress SIG_PIPE signal.

MSG_OOB = 0001

Receive flags: receive out-of-band data.

MSG_PEEK = \$0002

Receive flags: peek at data, don't remove from buffer.

MSG_PROXY = \$0010

Receive flags: ?

MSG_RST = \$1000

Receive flags: ?

MSG_SYN = \$0400

Receive flags: ?

MSG_TRUNC = \$0020

Receive flags: packet Data was discarded (buffer too small)

MSG_TRYHARD = MSG_DONTROUTE

Receive flags: ?

MSG_WAITALL = \$0100

Receive flags: Wait till operation completed.

NoAddress : in_addr = (s_addr: 0)

Constant indicating invalid (no) network address.

NoAddress6 : in6_addr = (u6_addr16: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0))

Constant indicating invalid (no) IPV6 network address.

NoNet : in_addr = (s_addr: 0)

Constant indicating invalid (no) network address.

NoNet6 : in6_addr = (u6_addr16: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0))

Constant indicating invalid (no) IPV6 network address.

PF_APPLETALK = AF_APPLETALK

Protocol family: Appletalk DDP

PF_ASH = AF_ASH

Protocol family: Ash

PF_ATMPVC = AF_ATMPVC

Protocol family: ATM PVCs

PF_ATMSVC = AF_ATMSVC

Protocol family: ATM SVCs

PF_AX25 = AF_AX25

Protocol family: Amateur Radio AX.25

PF_BLUETOOTH = AF_BLUETOOTH

Protocol family: Bluetooth sockets

PF_BRIDGE = AF_BRIDGE

Protocol family: Multiprotocol bridge

PF_DECnet = AF_DECnet

Protocol Family: DECNET project

PF_ECONET = AF_ECONET

Protocol family: Acorn Econet

PF_INET = AF_INET

Protocol family: Internet IP Protocol

PF_INET6 = AF_INET6

Protocol family: IP version 6

PF_IPX = AF_IPX

Protocol family: Novell IPX

PF_IRDA = AF_IRDA

Protocol family: IRDA sockets

PF_KEY = AF_KEY

Protocol family: Key management API

PF_LLC = AF_LLC

Protocol family: Linux LLC

PF_LOCAL = AF_LOCAL

Protocol family: Unix socket

PF_MAX = AF_MAX

Protocol family: Maximum value

PF_NETBEUI = AF_NETBEUI

Protocol family: Reserved for 802.2LLC project

PF_NETLINK = AF_NETLINK

Protocol family: ?

PF_NETROM = AF_NETROM

Protocol family:Amateur radio NetROM

PF_PACKET = AF_PACKET

Protocol family: Packet family

PF_PPPOX = AF_PPPOX

Protocol family: PPPoX sockets

PF_ROSE = AF_ROSE

Protocol family: Amateur Radio X.25 PLP

PF_ROUTE = AF_ROUTE

Protocol Family: ?

PF_SECURITY = AF_SECURITY

Protocol family: Security callback pseudo PF

PF_SNA = AF_SNA

Protocol Family: Linux SNA project

PF_TIPC = AF_TIPC

Protocol family: TIPC sockets

PF_UNIX = AF_UNIX

Protocol family: Unix domain sockets

PF_UNSPEC = AF_UNSPEC

Protocol family: Unspecified

PF_WANPIPE = AF_WANPIPE

Protocol family: Wanpipe API Sockets

PF_X25 = AF_X25

Protocol family: Reserved for X.25 project

SCM_SRCRT = IPV6_RXSRCRT

Undocumented Getsockopt option ?

SCM_TIMESTAMP = SO_TIMESTAMP

Socket option: ?

SHUT_RD = 0

Shutdown read part of full duplex socket

SHUT_RDWR = 2

Shutdown read and write part of full duplex socket

SHUT_WR = 1

Shutdown write part of full duplex socket

SOCK_DGRAM = 2

Type of socket: datagram (conn.less) socket (UDP)

SOCK_MAXADDRLLEN = 255

Maximum socket address length for Bind ([1058](#)) call.

SOCK_RAW = 3

Type of socket: raw socket

SOCK_RDM = 4

Type of socket: reliably-delivered message

SOCK_SEQPACKET = 5

Type of socket: sequential packet socket

SOCK_STREAM = 1

Type of socket: stream (connection) type socket (TCP)

SOL_ICMPV6 = 58

Socket level values for IPv6: ICMPV6

SOL_IP = 0

Undocumented ?

SOL_IPV6 = 41

Socket level values for IPv6: IPV6

SOL_SOCKET = 1

Socket option level: Socket level

SOMAXCONN = 128

Maximum queue length specifiable by listen.

SO_ACCEPTCONN = 30

Socket option: ?

SO_ATTACH_FILTER = 26

Socket option: ?

SO_BINDTODEVICE = 25

Socket option: ?

SO_BROADCAST = 6

Socket option: Broadcast

SO_BSDCOMPAT = 14

Socket option: ?

SO_DEBUG = 1

Socket option level: debug

SO_DETACH_FILTER = 27

Socket option: ?

SO_DONTROUTE = 5

Socket option: Don't route

SO_ERROR = 4

Socket option: Error

SO_KEEPALIVE = 9

Socket option: keep alive

SO_LINGER = 13

Socket option: ?

SO_NO_CHECK = 11

Socket option: ?

SO_OOBINLINE = 10

Socket option: ?

SO_PASSCRED = 16

Socket option: ?

SO_PEERCRECRED = 17

Socket option: ?

SO_PEERNAME = 28

Socket option: ?

SO_PRIORITY = 12

Socket option: ?

SO_RCVBUF = 8

Socket option: receive buffer

SO_RCVLOWAT = 18

Socket option: ?

SO_RCVTIMEO = 20

Socket option: ?

SO_REUSEADDR = 2

Socket option: Reuse address

SO_SECURITY_AUTHENTICATION = 22

Socket option: ?

SO_SECURITY_ENCRYPTION_NETWORK = 24

Socket option: ?

SO_SECURITY_ENCRYPTION_TRANSPORT = 23

Socket option: ?

SO_SNDBUF = 7

Socket option: Send buffer

SO_SNDLOWAT = 19

Socket option: ?

SO_SNDTIMEO = 21

Socket option: ?

SO_TIMESTAMP = 29

Socket option: ?

SO_TYPE = 3

Socket option: Type

S_IN = 0

Input socket in socket pair.

S_OUT = 1

Output socket in socket pair

TCP_CONGESTION = 13

Get/set the congestion-control algorithm for this socket

TCP_CORK = 3

Get/Set CORK algorithm: Send only complete packets

TCP_DEFER_ACCEPT = 9

Get/Set deferred accept on server socket

TCP_INFO = 11

Get TCP connection information (linux only)

TCP_KEEPCNT = 6

Get/Set retry count for unacknowledged KEEPALIVE transmissions.

TCP_KEEPIDL = 4

Get/Set inactivity interval between KEEPALIVE transmissions.

TCP_KEEPINTVL = 5

Get/Set retry interval for unacknowledged KEEPALIVE transmissions.

TCP_LINGER2 = 8

Get/Set Linger2 flag

TCP_MAXSEG = 2

Get/Set Maximum segment size

TCP_MD5SIG = 14

Get/Set TCP MD5 signature option

TCP_NODELAY = 1

Get/Set No delay flag: disable Nagle algorithm

TCP_QUICKACK = 12

Get/Set quick ACK packet option.

TCP_SYNCNT = 7

Get/Set number of SYN packets to send before giving up on connection establishment

TCP_WINDOW_CLAMP = 10

Get/Set maximum packet size

UDP_CORK = 1

Get/Set UDP CORK algorithm on datagram sockets

UDP_ENCAP = 100

Get/Set UDP encapsulation flag for IPSec datagram sockets

UDP_ENCAP_ESPINUDP = 2

? Undocumented datagram option, IPSec related

UDP_ENCAP_ESPINUDP_NON_IKE = 1

? Undocumented datagram option, IPSec related

UDP_ENCAP_L2TPINUDP = 3

? Undocumented datagram option, IPSec related

33.3.2 Types

```
in6_addr = packed record
end
```

Record used to describe a general IPV6 address.

```
in_addr = packed record
end
```

General inet socket address.

```
linger = packed record
  l_onoff : cint;
  l_linger : cint;
end
```

This record is used in the `fpsetsockopt` ([1067](#)) call to specify linger options.

PIn6Addr = pin6_addr

Pointer to in6_addr ([1053](#)) type.

pin6_addr = ^in6_addr

Pointer to Tin6_addr ([1055](#))

PInAddr = pin_addr

Alias for pin_addr ([1054](#))

PInetSockAddr = psockaddr_in

Pointer to sockaddr_in (1054)

```
PInetSockAddr6 = psockaddr_in6
```

Pointer to sockaddr_in6 (1055) type

```
pin_addr = ^in_addr
```

Pointer to in_addr (1053) record.

```
plinger = ^linger
```

Pointer to linger (1053) type.

```
psockaddr = ^sockaddr
```

Pointer to TSocketAddr (1055)

```
psockaddr_in = ^sockaddr_in
```

Pointer to sockaddr_in (1054)

```
psockaddr_in6 = ^sockaddr_in6
```

Pointer to sockaddr_in6 (1055)

```
psockaddr_un = ^sockaddr_un
```

Pointer to sockaddr_un (1055) type.

```
sa_family_t = cushort
```

Address family type

```
sockaddr = packed record
end
```

sockaddr is used to store a general socket address for the FPBind (1061), FPrece (1065) and FPSend (1066) calls.

```
sockaddr_in = packed record
  sin_family : sa_family_t;
  sin_port   : cushort;
  sin_addr   : in_addr;
  xpad       : Array[0..7] of Char;
end
```

sockaddr_in is used to store a INET socket address for the FPBind (1061), FPrece (1065) and FPSend (1066) calls.

```

sockaddr_in6 = packed record
  sin6_family : sa_family_t;
  sin6_port   : cuint16;
  sin6_flowinfo : cuint32;
  sin6_addr   : in6_addr;
  sin6_scope_id : cuint32;
end

```

Alias for `sockaddr_in6` (1055)

```

sockaddr_un = packed record
  sun_family : sa_family_t;
  sun_path   : Array[0..107] of Char;
end

```

`sockaddr_un` is used to store a UNIX socket address for the `FPBind` (1061), `FPRecv` (1065) and `FPSend` (1066) calls.

```
TIn6Addr = in6_addr
```

Alias for `in6_addr` (1053) type.

```
Tin6_addr = in6_addr
```

Alias for `sockaddr_in6` (1055)

```
TInAddr = in_addr
```

Alias for `in_addr` (1053) record type.

```
TInetSockAddr = sockaddr_in
```

Alias for `sockaddr_in` (1054)

```
TInetSockAddr6 = sockaddr_in6
```

Alias for `sockaddr_in6` (1055)

```
TIn_addr = in_addr
```

Alias for `in_addr` (1053) record type.

```
TLinger = linger
```

Alias for `linger` (1053)

```
TSockAddr = sockaddr
```

```
TSockArray = Array[1..2] of LongInt
```

Type returned by the `FPSocketPair` (1069) call.

```
Tsocket = LongInt
```

Alias for easy kylix porting

```
TSockLen = BaseUnix.TSocklen
```

The actual type of `TSockLen` depends on the platform.

```
TSockPairArray = Array[0..1] of LongInt
```

Array of sockets, used in `FPSocketPair` (1069) call.

```
TUnixSockAddr = packed record
    family : sa_family_t;
    path : Array[0..107] of Char;
end
```

Alias for `sockaddr_un` (1055)

33.4 Procedures and functions

33.4.1 Accept

Synopsis: Accept a connection from a socket (deprecated).

```
Declaration: function Accept(Sock: LongInt; var addr: TInetSockAddr; var SockIn: File;
    var SockOut: File) : Boolean
function Accept(Sock: LongInt; var addr: TInetSockAddr; var SockIn: text;
    var SockOut: text) : Boolean
function Accept(Sock: LongInt; var addr: string; var SockIn: text;
    var SockOut: text) : Boolean
function Accept(Sock: LongInt; var addr: string; var SockIn: File;
    var SockOut: File) : Boolean
```

Visibility: default

Description: `Accept` accepts a connection from a socket `Sock`, which was listening for a connection. If a connection is accepted, a file descriptor is returned. On error `-1` is returned. The returned socket may NOT be used to accept more connections. The original socket remains open.

The `Accept` call fills the address of the connecting entity in `Addr`, and sets its length in `AddrLen`. `Addr` should be pointing to enough space, and `AddrLen` should be set to the amount of space available, prior to the call.

The alternate forms of the `Accept` (1056) command, with the `Text` or `File` parameters are equivalent to subsequently calling the regular `Accept` (1056) function and the `Sock2Text` (1072) or `Sock2File` (1072) functions. These functions return `True` if successful, `False` otherwise.

Errors: On error, `-1` is returned, and errors are reported in `SocketError`, and include the following:

ESockEBADF (1035) The socket descriptor is invalid.

ESockENOTSOCK (1035)The descriptor is not a socket.

SYS_EOPNOTSUPPThe socket type doesn't support the `Listen` operation.

ESockEFAULT (1035)`Addr` points outside your address space.

ESockEWOULDBLOCK (1035)The requested operation would block the process.

See also: `FPListen` (1065), `Connect` (1058), `FPConnect` (1062), `FPBind` (1061)

Listing: `./sockex/socksvr.pp`

Program `server`;

```
{
  Program to test Sockets unit by Michael van Canneyt and Peter Vreman
  Server Version, First Run sock_svr to let it create a socket and then
  sock_cli to connect to that socket
}
{$mode fpc}
uses Sockets;

Var
  FromName : string;
  Buffer    : string[255];
  S         : Longint;
  Sin, Sout : Text;
  SAddr     : TInetSockAddr;

procedure perror (const S:string);
begin
  writeln (S, SocketError);
  halt(100);
end;

begin
  S:=fpSocket (AF_INET, SOCK_STREAM, 0);
  if SocketError<>0 then
    PError ('Server : Socket : ');
  SAddr.sin_family:=AF_INET;
  { port 50000 in network order }
  SAddr.sin_port:=htons(5000);
  SAddr.sin_addr.s_addr:=0;
  if fpBind(S, @SAddr, sizeof(saddr))=-1 then
    PError ('Server : Bind : ');
  if fpListen (S, 1)=-1 then
    PError ('Server : Listen : ');
  Writeln('Waiting for Connect from Client, run now sock_cli in an other tty');
  if Accept(S, FromName, Sin, Sout) then
    PError ('Server : Accept : '+fromname);
  Reset(Sin);
  Rewrite(Sout);
  Writeln(Sout, 'Message From Server');
  Flush(SOut);
  while not eof(sin) do
    begin
      Readln(Sin, Buffer);
      Writeln('Server : read : ', buffer);
    end;
end.
```

33.4.2 Bind

Synopsis: Alias for `fpBind`.

Declaration: `function Bind(Sock: LongInt; const addr: string) : Boolean`

Visibility: default

Description: `Bind` is an alias for `fpBind` (1061) which binds to either a socket struct (`Addr`) or a string indicating a unix socket file (unix socket).

This function is deprecated, use `fpBind` instead.

See also: `FPBind` (1061)

33.4.3 CloseSocket

Synopsis: Closes a socket handle.

Declaration: `function CloseSocket(Sock: LongInt) : LongInt`

Visibility: default

Description: `CloseSocket` closes a socket handle. It returns 0 if the socket was closed successfully, -1 if it failed.

Errors: On error, -1 is returned.

See also: `FPSocket` (1068)

33.4.4 Connect

Synopsis: Open a connection to a server socket (deprecated).

Declaration: `function Connect(Sock: LongInt; const addr: TInetSockAddr;
 var SockIn: text; var SockOut: text) : Boolean
 function Connect(Sock: LongInt; const addr: TInetSockAddr;
 var SockIn: File; var SockOut: File) : Boolean
 function Connect(Sock: LongInt; const addr: string; var SockIn: text;
 var SockOut: text) : Boolean
 function Connect(Sock: LongInt; const addr: string; var SockIn: File;
 var SockOut: File) : Boolean`

Visibility: default

Description: `Connect` opens a connection to a peer, whose address is described by `Addr`. `AddrLen` contains the length of the address. The type of `Addr` depends on the kind of connection you're trying to make, but is generally one of `TSockAddr` or `TUnixSockAddr`.

The forms of the `Connect` (1058) command with the `Text` or `File` arguments are equivalent to subsequently calling the regular `Connect` function and the `Sock2Text` (1072) or `Sock2File` (1072) functions. These functions return `True` if successful, `False` otherwise.

The `Connect` function returns a file descriptor if the call was successful, -1 in case of error.

Errors: On error, -1 is returned and errors are reported in `SocketError`.

See also: `FPListen` (1065), `FPBind` (1061), `Accept` (1056), `FPAccept` (1060)

Listing: `./sockex/sockcli.pp`

Program Client;

```
{
  Program to test Sockets unit by Michael van Canneyt and Peter Vreman
  Client Version, First Run sock_svr to let it create a socket and then
  sock_cli to connect to that socket
}
```

uses Sockets;

```
procedure PError(const S : string);
begin
  writeln(S, SocketError);
  halt(100);
end;
```

Var

```
SAddr    : TInetSockAddr;
Buffer    : string [255];
S         : Longint;
Sin, Sout : Text;
i         : integer;
```

begin

```
S:=fpSocket (AF_INET, SOCK_STREAM, 0);
if s=-1 then
  PError('Client : Socket : ');
SAddr.sin_family:=AF_INET;
{ port 50000 in network order }
SAddr.sin_port:=htons(5000);
{ localhost : 127.0.0.1 in network order }
SAddr.sin_addr.s_addr:=HostToNet((127 shl 24) or 1);
if not Connect (S, SAddr, Sin, Sout) then
  PError('Client : Connect : ');
Reset(Sin);
ReWrite(Sout);
Buffer:='This is a textstring sent by the Client.';
for i:=1 to 10 do
  Writeln(Sout, Buffer);
Flush(Sout);
Readln(Sin, Buffer);
WriteLn(Buffer);
Close(sout);
```

end.

Listing: ./sockex/pfinger.pp

program pfinger;

uses sockets, errors;

Var

```
Addr : TInetSockAddr;
S : Longint;
Sin, Sout : Text;
Line : string;
```

```

begin
  Addr.sin_family:=AF_INET;
  { port 79 in network order }
  Addr.sin_port:=79 shl 8;
  { localhost : 127.0.0.1 in network order }
  Addr.sin_addr.s_addr:=((1 shl 24) or 127);
  S:=fpSocket(AF_INET,SOCK_STREAM,0);
  If Not Connect (S,ADDR,SIN,SOUT) Then
    begin
      Writeln ('Couldn't connect to localhost');
      Writeln ('Socket error : ',strerror(SocketError));
      halt(1);
    end;
  rewrite (sout);
  reset(sin);
  writeln (sout,paramstr(1));
  flush(sout);
  while not eof(sin) do
    begin
      readln (Sin,line);
      writeln (line);
    end;
  fpShutdown(s,2);
  close (sin);
  close (sout);
end.

```

33.4.5 fpaccept

Synopsis: Accept a connection from a socket.

Declaration: `function fpaccept(s: cint;addrx: psockaddr;addrlen: pSockLen) : cint`

Visibility: default

Description: `Accept` accepts a connection from a socket `S`, which was listening for a connection. If a connection is accepted, a file descriptor is returned (positive number). On error `-1` is returned. The returned socket may NOT be used to accept more connections. The original socket remains open.

The `Accept` call fills the address of the connecting entity in `Addrx`, and sets its length in `Addrlen`. `Addrx` should be pointing to enough space, and `Addrlen` should be set to the amount of space available, prior to the call.

Errors: On error, `-1` is returned, and errors are reported in `SocketError`, and include the following:

ESockEBADF (1035)The socket descriptor is invalid.

ESockENOTSOCK (1035)The descriptor is not a socket.

SYS_EOPNOTSUPPThe socket type doesn't support the `Listen` operation.

ESockEFAULT (1035)`Addr` points outside your address space.

ESockEWOULDBLOCK (1035)The requested operation would block the process.

See also: `fpListen` (1065), `fpConnect` (1062), `fpBind` (1061)

Listing: `./sockex/socksvr.pp`

```

Program server;

{
  Program to test Sockets unit by Michael van Canneyt and Peter Vreman
  Server Version, First Run sock_svr to let it create a socket and then
  sock_cli to connect to that socket
}
{$mode fpc}
uses Sockets;

Var
  FromName : string;
  Buffer    : string[255];
  S         : Longint;
  Sin, Sout : Text;
  SAddr     : TInetSockAddr;

procedure perror (const S:string);
begin
  writeln (S, SocketError);
  halt(100);
end;

begin
  S:=fpSocket (AF_INET, SOCK_STREAM, 0);
  if SocketError<>0 then
    Perror ('Server : Socket : ');
  SAddr.sin_family:=AF_INET;
  { port 50000 in network order }
  SAddr.sin_port:=htons(5000);
  SAddr.sin_addr.s_addr:=0;
  if fpBind(S, @SAddr, sizeof(saddr))=-1 then
    PError ('Server : Bind : ');
  if fpListen (S, 1)=-1 then
    PError ('Server : Listen : ');
  Writeln ('Waiting for Connect from Client , run now sock_cli in an other tty');
  if Accept(S, FromName, Sin, Sout) then
    PError ('Server : Accept : '+fromname);
  Reset(Sin);
  ReWrite(Sout);
  Writeln (Sout, 'Message From Server');
  Flush(SOut);
  while not eof(sin) do
    begin
      Readln(Sin, Buffer);
      Writeln('Server : read : ', buffer);
    end;
end.

```

33.4.6 fpbind

Synopsis: Bind a socket to an address.

Declaration: `function fpbind(s: cint; addrx: psockaddr; addrlen: TSockLen) : cint`

Visibility: default

Description: `fpBind` binds the socket `s` to address `Addrx`. `Addrx` has length `AddrLen`. The function returns 0 if the call was succesful, -1 if not.

Errors: Errors are returned in `SocketError` and include the following:

ESockEBADF (1035)The socket descriptor is invalid.

ESockEINVAL (1035)The socket is already bound to an address,

ESockEACCESS (1035)Address is protected and you don't have permission to open it.

More errors can be found in the Unix man pages.

See also: `FPSocket` (1068)

33.4.7 fpconnect

Synopsis: Open a connection to a server socket.

Declaration: `function fpconnect(s: cint;name: psockaddr;namelen: TSockLen) : cint`

Visibility: default

Description: `fpConnect` uses the socket `s` to open a connection to a peer, whose address is described by `Name`. `NameLen` contains the length of the address. The type of `Name` depends on the kind of connection you are trying to make, but is generally one of `TSockAddr` or `TUnixSockAddr`.

The `fpConnect` function returns zero if the call was successfull, -1 in case of error.

Errors: On error, -1 is returned and errors are reported in `SocketError`.

See also: `fpListen` (1065), `fpBind` (1061), `fpAccept` (1060)

Listing: `./sockex/sockcli.pp`

Program Client;

```
{
  Program to test Sockets unit by Michael van Canneyt and Peter Vreman
  Client Version , First Run sock_svr to let it create a socket and then
  sock_cli to connect to that socket
}
```

uses Sockets;

```
procedure PError(const S : string);
begin
  writeln(S, SocketError);
  halt(100);
end;
```

Var

```
SAddr    : TInetSockAddr;
Buffer   : string [255];
S        : Longint;
Sin, Sout : Text;
i        : integer;
```

begin

```

S:=fpSocket (AF_INET,SOCK_STREAM,0);
if s=-1 then
  Perror('Client : Socket : ');
SAddr.sin_family:=AF_INET;
{ port 50000 in network order }
SAddr.sin_port:=htons(5000);
{ localhost : 127.0.0.1 in network order }
SAddr.sin_addr.s_addr:=HostToNet((127 shl 24) or 1);
if not Connect (S,SAddr,Sin,Sout) then
  PError('Client : Connect : ');
Reset(Sin);
ReWrite(Sout);
Buffer:='This is a textstring sent by the Client.';
for i:=1 to 10 do
  Writeln(Sout, Buffer);
Flush(Sout);
Readln(Sin, Buffer);
WriteLn( Buffer);
Close(sout);
end.

```

Listing: ./sockex/pfinger.pp

```

program pfinger;

uses sockets , errors;

Var
  Addr : TInetSockAddr;
  S : Longint;
  Sin, Sout : Text;
  Line : string;

begin
  Addr.sin_family:=AF_INET;
  { port 79 in network order }
  Addr.sin_port:=79 shl 8;
  { localhost : 127.0.0.1 in network order }
  Addr.sin_addr.s_addr:=((1 shl 24) or 127);
  S:=fpSocket(AF_INET,SOCK_STREAM,0);
  If Not Connect (S,ADDR,SIN,SOUT) Then
    begin
      Writeln ( 'Couldn't connect to localhost');
      Writeln ( 'Socket error : ',strerror(SocketError));
      halt(1);
    end;
  rewrite (sout);
  reset(sin);
  writeln (sout, paramstr(1));
  flush(sout);
  while not eof(sin) do
    begin
      readln (Sin, line);
      writeln (line);
    end;
  fpShutdown(s,2);
  close (sin);
  close (sout);
end.

```

33.4.8 `fpgetpeername`

Synopsis: Return the name (address) of the connected peer.

Declaration: `function fpgetpeername(s: cint; name: psockaddr; namelen: pSockLen) : cint`

Visibility: default

Description: `fpGetPeerName` returns the name of the entity connected to the specified socket `S`. The Socket must be connected for this call to work.

Name should point to enough space to store the name, the amount of space pointed to should be set in `Namelen`. When the function returns successfully, Name will be filled with the name, and Name will be set to the length of Name.

Errors: Errors are reported in `SocketError`, and include the following:

ESockEBADF (1035)The socket descriptor is invalid.

ESockENOBUFS (1035)The system doesn't have enough buffers to perform the operation.

ESockENOTSOCK (1035)The descriptor is not a socket.

ESockEFAULT (1035)`Addr` points outside your address space.

ESockENOTCONN (1035)The socket isn't connected.

See also: `fpConnect` (1062), `fpSocket` (1068)

33.4.9 `fpgetsockname`

Synopsis: Return name of socket.

Declaration: `function fpgetsockname(s: cint; name: psockaddr; namelen: pSockLen) : cint`

Visibility: default

Description: `fpGetSockName` returns the current name of the specified socket `S`. Name should point to enough space to store the name, the amount of space pointed to should be set in `Namelen`. When the function returns successfully, Name will be filled with the name, and `Namelen` will be set to the length of Name.

Errors: Errors are reported in `SocketError`, and include the following:

ESockEBADF (1035)The socket descriptor is invalid.

ESockENOBUFS (1035)The system doesn't have enough buffers to perform the operation.

ESockENOTSOCK (1035)The descriptor is not a socket.

ESockEFAULT (1035)`Addr` points outside your address space.

See also: `fpBind` (1061)

33.4.10 fpgetsockopt

Synopsis: Get current socket options

Declaration: `function fpgetsockopt(s: cint; level: cint; optname: cint; optval: pointer; optlen: pSockLen) : cint`

Visibility: default

Description: `fpGetSockOpt` gets the connection option `optname`, for socket `S`. The socket may be obtained from different levels, indicated by `Level`, which can be one of the following:

SOL_SOCKET From the socket itself.

XXX set `Level` to `XXX`, the protocol number of the protocol which should interpret the option.

The options are stored in the memory location pointed to by `optval`. `optlen` should point to the initial length of `optval`, and on return will contain the actual length of the stored data.

On success, 0 is returned. On Error, -1 is returned.

Errors: Errors are reported in `SocketError`, and include the following:

ESockEBADF (1035) The socket descriptor is invalid.

ESockENOTSOCK (1035) The descriptor is not a socket.

ESockEFAULT (1035) `OptVal` points outside your address space.

See also: `fpSetSockOpt` (1067)

33.4.11 fplisten

Synopsis: Listen for connections on a socket.

Declaration: `function fplisten(s: cint; backlog: cint) : cint`

Visibility: default

Description: `fpListen` listens for up to `backlog` connections from socket `S`. The socket `S` must be of type `SOCK_STREAM` or `Sock_SEQPACKET`.

The function returns 0 if a connection was accepted, -1 if an error occurred.

Errors: Errors are reported in `SocketError`, and include the following:

ESockEBADF (1035) The socket descriptor is invalid.

ESockENOTSOCK (1035) The descriptor is not a socket.

SYS_EOPNOTSUPP The socket type doesn't support the `Listen` operation.

See also: `fpSocket` (1068), `fpBind` (1061), `fpConnect` (1062)

33.4.12 fprecv

Synopsis: Receive data on socket

Declaration: `function fprecv(s: cint; buf: pointer; len: size_t; flags: cint) : ssize_t`

Visibility: default

Description: `fpRecv` reads at most `len` bytes from socket `S` into address `buf`. The socket must be in a connected state. `Flags` can be one of the following:

- 1**Process out-of band data.
- 4**Bypass routing, use a direct interface.
- ??**Wait for full request or report an error.

The functions returns the number of bytes actually read from the socket, or -1 if a detectable error occurred.

Errors: Errors are reported in `SocketError`, and include the following:

- ESockEBADF (1035)**The socket descriptor is invalid.
- ESockENOTCONN (1035)**The socket isn't connected.
- ESockENOTSOCK (1035)**The descriptor is not a socket.
- ESockEFAULT (1035)**The address is outside your address space.
- ESockEMSGSIZE (1035)**The message cannot be sent atomically.
- ESockEWOULDBLOCK (1035)**The requested operation would block the process.
- ESockENOBUFS (1035)**The system doesn't have enough free buffers available.

See also: `FPSend` (1066)

33.4.13 `fprecvfrom`

Synopsis: Receive data from an unconnected socket

Declaration: `function fprecvfrom(s: cint;buf: pointer;len: size_t;flags: cint;
from: psockaddr;fromlen: pSockLen) : ssize_t`

Visibility: default

Description: `fpRecvFrom` receives data in buffer `Buf` with maximum length `Len` from socket `S`. Receipt is controlled by options in `Flags`. The location pointed to by `from` will be filled with the address from the sender, and it's length will be stored in `fromlen`. The function returns the number of bytes received, or -1 on error. `AddrLen`.

Errors: On error, -1 is returned.

See also: `fpSocket` (1068), `frecv` (1065)

33.4.14 `fpsend`

Synopsis: Send data through socket

Declaration: `function fpsend(s: cint;msg: pointer;len: size_t;flags: cint) : ssize_t`

Visibility: default

Description: `fpSend` sends `Len` bytes starting from address `Msg` to socket `S`. `S` must be in a connected state. Options can be passed in `Flags`.

The function returns the number of bytes sent, or -1 if a detectable error occurred.

`Flags` can be one of the following:

- 1**Process out-of band data.

4Bypass routing, use a direct interface.

Errors: Errors are reported in `SocketError`, and include the following:

- ESockEBADF (1035)**The socket descriptor is invalid.
- ESockENOTSOCK (1035)**The descriptor is not a socket.
- ESockEFAULT (1035)**The address is outside your address space.
- ESockEMSGSIZE (1035)**The message cannot be sent atomically.
- ESockEWOULDBLOCK (1035)**The requested operation would block the process.
- ESockENOBUFS (1035)**The system doesn't have enough free buffers available.

See also: `fpRecv` (1065)

33.4.15 `fpsendto`

Synopsis: Send data through an unconnected socket to an address.

Declaration: `function fpsendto(s: cint;msg: pointer;len: size_t;flags: cint;
 tox: psockaddr;tolen: TSocketLen) : ssize_t`

Visibility: default

Description: `fpSendTo` sends data from buffer `Msg` with length `len` through socket `S` with options `Flags`.
The data is sent to address `tox`, which has length `toLen`

Errors: On error, -1 is returned.

See also: `fpSocket` (1068), `fpSend` (1066), `fpRecvFrom` (1066)

33.4.16 `fpsetsockopt`

Synopsis: Set socket options.

Declaration: `function fpsetsockopt(s: cint;level: cint;optname: cint;optval: pointer;
 optlen: TSocketLen) : cint`

Visibility: default

Description: `fpSetSockOpt` sets the connection options for socket `S`. The socket may be manipulated at different levels, indicated by `Level`, which can be one of the following:

- SOL_SOCKET**To manipulate the socket itself.
- XXX**set `Level` to `XXX`, the protocol number of the protocol which should interpret the option.

The actual option is stored in a buffer pointed to by `optval`, with length `optlen`.

For more information on this call, refer to the unix manual page `setsockopt`

Errors: Errors are reported in `SocketError`, and include the following:

- ESockEBADF (1035)**The socket descriptor is invalid.
- ESockENOTSOCK (1035)**The descriptor is not a socket.
- ESockEFAULT (1035)**`OptVal` points outside your address space.

See also: `fpGetSockOpt` (1065)

33.4.17 fpshutdown

Synopsis: Close one end of full duplex connection.

Declaration: `function fpshutdown(s: cint;how: cint) : cint`

Visibility: default

Description: `fpShutDown` closes one end of a full duplex socket connection, described by `S`. The parameter `How` determines how the connection will be shut down, and can be one of the following:

0Further receives are disallowed.

1Further sends are disallowed.

2Sending nor receiving are allowed.

On succes, the function returns 0, on error -1 is returned.

Errors: `SocketError` is used to report errors, and includes the following:

ESockEBADF (1035)The socket descriptor is invalid.

ESockENOTCONN (1035)The socket isn't connected.

ESockENOTSOCK (1035)The descriptor is not a socket.

See also: `fpSocket` (1068), `fpConnect` (1062)

33.4.18 fpsocket

Synopsis: Create new socket

Declaration: `function fpsocket(domain: cint;xtype: cint;protocol: cint) : cint`

Visibility: default

Description: `fpSocket` creates a new socket in domain `Domain`, from type `xType` using protocol `Protocol`. The `Domain`, `Socket` type and `Protocol` can be specified using predefined constants (see the section on constants for available constants) If succesfull, the function returns a socket descriptor, which can be passed to a subsequent `fpBind` (1061) call. If unsuccesfull, the function returns -1.

for an example, see `Accept` (1056).

Errors: Errors are returned in `SocketError`, and include the follwing:

ESockEPROTONOSUPPORT (1035)The protocol type or the specified protocol is not supported within this domain.

ESockEMFILE (1035)The per-process descriptor table is full.

SYS_ENFILEThe system file table is full.

ESockEACCESS (1035)Permission to create a socket of the specified type and/or protocol is denied.

ESockENOBUFS (1035)Insufficient buffer space is available. The socket cannot be created until sufficient resources are freed.

See also: `FPSocketPair` (1069)

33.4.19 fpsocketpair

Synopsis: Create socket pair.

Declaration: `function fpsocketpair(d: cint; xtype: cint; protocol: cint; sv: puint) : cint`

Visibility: default

Description: `fpSocketPair` creates 2 sockets in domain `D`, from type `xType` and using protocol `Protocol`. The pair is returned in `sv`, and they are indistinguishable. The function returns -1 upon error and 0 upon success.

Errors: Errors are reported in `SocketError`, and are the same as in `FPocket` ([1068](#))

See also: `Str2UnixSockAddr` ([1073](#))

33.4.20 HostAddrToStr

Synopsis: Convert a host address to a string.

Declaration: `function HostAddrToStr(Entry: in_addr) : AnsiString`

Visibility: default

Description: `HostAddrToStr` converts the host address in `Entry` to a string representation in human-readable form (a dotted quad).

Basically, it is the same as `NetAddrToStr` ([1070](#)), but with the bytes in correct order.

See also: `NetAddrToStr` ([1070](#)), `StrToHostAddr` ([1073](#)), `StrToNetAddr` ([1073](#))

33.4.21 HostAddrToStr6

Synopsis: Convert a IPV6 host address to a string representation.

Declaration: `function HostAddrToStr6(Entry: Tin6_addr) : AnsiString`

Visibility: default

Description: `HostAddrToStr6` converts the IPV6 host address in `Entry` to a string representation in human-readable form.

Basically, it is the same as `NetAddrToStr6` ([1070](#)), but with the bytes in correct order.

See also: `NetAddrToStr` ([1070](#)), `StrToHostAddr` ([1073](#)), `StrToNetAddr` ([1073](#)), `StrToHostAddr6` ([1073](#))

33.4.22 HostToNet

Synopsis: Convert a host address to a network address

Declaration: `function HostToNet(Host: in_addr) : in_addr`
`function HostToNet(Host: LongInt) : LongInt`

Visibility: default

Description: `HostToNet` converts a host address to a network address. It takes care of endianness of the host machine. The address can be specified as a dotted quad or as a longint.

Errors: None.

See also: `NetToHost` ([1071](#)), `NToHS` ([1071](#)), `HTONS` ([1070](#)), `ShortHostToNet` ([1071](#)), `ShortNetToHost` ([1072](#))

33.4.23 htonl

Synopsis: Convert long integer from host ordered to network ordered

Declaration: `function htonl(host: Cardinal) : Cardinal; Overload`

Visibility: default

Description: `htonl` makes sure that the bytes in `host` are ordered in the correct way for sending over the network and returns the correctly ordered result.

See also: `htons` ([1070](#)), `ntohl` ([1071](#)), `ntohs` ([1071](#))

33.4.24 htons

Synopsis: Convert short integer from host ordered to network ordered

Declaration: `function htons(host: Word) : Word`

Visibility: default

Description: `htons` makes sure that the bytes in `host` are ordered in the correct way for sending over the network and returns the correctly ordered result.

See also: `htonl` ([1070](#)), `ntohl` ([1071](#)), `ntohs` ([1071](#))

33.4.25 NetAddrToStr

Synopsis: Convert a network address to a string.

Declaration: `function NetAddrToStr(Entry: in_addr) : AnsiString`

Visibility: default

Description: `NetAddrToStr` converts the network address in `Entry` to a string representation in human-readable form (a dotted quad).

See also: `HostAddrToStr` ([1069](#)), `StrToNetAddr` ([1073](#)), `StrToHostAddr` ([1073](#))

33.4.26 NetAddrToStr6

Synopsis: Convert a IPV6 network address to a string.

Declaration: `function NetAddrToStr6(Entry: Tin6_addr) : AnsiString`

Visibility: default

Description: `NetAddrToStr6` converts the IPV6 network address in `Entry` to a string representation in human-readable form.

Basically, it is the same as `NetAddrToStr6` ([1070](#)), but with the bytes in correct order.

See also: `NetAddrToStr` ([1070](#)), `StrToHostAddr` ([1073](#)), `StrToNetAddr` ([1073](#)), `StrToHostAddr6` ([1073](#))

33.4.27 NetToHost

Synopsis: Convert a network address to a host address.

Declaration: `function NetToHost (Net: in_addr) : in_addr`
`function NetToHost (Net: LongInt) : LongInt`

Visibility: default

Description: `NetToHost` converts a network address to a host address. It takes care of endianness of the host machine. The address can be specified as a dotted quad or as a longint.

Errors: None.

See also: `HostToNet` ([1069](#)), `NToHS` ([1071](#)), `HToNS` ([1070](#)), `ShortHostToNet` ([1071](#)), `ShortNetToHost` ([1072](#))

33.4.28 NToHI

Synopsis: Convert long integer from network ordered to host ordered

Declaration: `function NToHI (Net: Cardinal) : Cardinal; Overload`

Visibility: default

Description: `ntohs` makes sure that the bytes in `Net`, received from the network, are ordered in the correct way for handling by the host machine, and returns the correctly ordered result.

See also: `htonl` ([1070](#)), `htons` ([1070](#)), `ntohs` ([1071](#))

33.4.29 NToHs

Synopsis: Convert short integer from network ordered to host ordered

Declaration: `function NToHs (Net: Word) : Word`

Visibility: default

Description: `ntohs` makes sure that the bytes in `Net`, received from the network, are ordered in the correct way for handling by the host machine, and returns the correctly ordered result.

See also: `htonl` ([1070](#)), `htons` ([1070](#)), `ntohl` ([1071](#))

33.4.30 ShortHostToNet

Synopsis: Convert a host port number to a network port number

Declaration: `function ShortHostToNet (Host: Word) : Word`

Visibility: default

Description: `ShortHostToNet` converts a host port number to a network port number. It takes care of endianness of the host machine.

Errors: None.

See also: `ShortNetToHost` ([1072](#)), `HostToNet` ([1069](#)), `NToHS` ([1071](#)), `HToNS` ([1070](#))

33.4.31 ShortNetToHost

Synopsis: Convert a network port number to a host port number

Declaration: `function ShortNetToHost (Net: Word) : Word`

Visibility: default

Description: `ShortNetToHost` converts a network port number to a host port number. It takes care of endianness of the host machine.

Errors: None.

See also: `ShortNetToHost` ([1072](#)), `HostToNet` ([1069](#)), `NToHS` ([1071](#)), `HToNS` ([1070](#))

33.4.32 Sock2File

Synopsis: Convert socket to untyped file descriptors

Declaration: `procedure Sock2File (Sock: LongInt; var SockIn: File; var SockOut: File)`

Visibility: default

Description: `Sock2File` transforms a socket `Sock` into 2 Pascal file descriptors of type `File`, one for reading from the socket (`SockIn`), one for writing to the socket (`SockOut`).

Errors: None.

See also: `FPSocket` ([1068](#)), `Sock2Text` ([1072](#))

33.4.33 Sock2Text

Synopsis: Convert socket to text file descriptors

Declaration: `procedure Sock2Text (Sock: LongInt; var SockIn: Text; var SockOut: Text)`

Visibility: default

Description: `Sock2Text` transforms a socket `Sock` into 2 Pascal file descriptors of type `Text`, one for reading from the socket (`SockIn`), one for writing to the socket (`SockOut`).

Errors: None.

See also: `FPSocket` ([1068](#)), `Sock2File` ([1072](#))

33.4.34 socketerror

Synopsis: Contains the error code for the last socket operation.

Declaration: `function socketerror : cint`

Visibility: default

Description: `SocketError` contains the error code for the last socket operation. It can be examined to return the last socket error.

33.4.35 Str2UnixSockAddr

Synopsis: Convert path to TUnixSockAddr ([1056](#))

Declaration: `procedure Str2UnixSockAddr(const addr: string; var t: TUnixSockAddr;
var len: LongInt)`

Visibility: default

Description: `Str2UnixSockAddr` transforms a Unix socket address in a string to a `TUnixSockAddr` structure which can be passed to the `Bind` ([1058](#)) call.

Errors: None.

See also: `FPocket` ([1068](#)), `FPBind` ([1061](#))

33.4.36 StrToHostAddr

Synopsis: Convert a string to a host address.

Declaration: `function StrToHostAddr(IP: AnsiString) : in_addr`

Visibility: default

Description: `StrToHostAddr` converts the string representation in `IP` to a host address and returns the host address.

Errors: On error, the host address is filled with zeroes.

See also: `NetAddrToStr` ([1070](#)), `HostAddrToStr` ([1069](#)), `StrToNetAddr` ([1073](#))

33.4.37 StrToHostAddr6

Synopsis: Convert a string to a IPV6 host address.

Declaration: `function StrToHostAddr6(IP: string) : Tin6_addr`

Visibility: default

Description: `StrToHostAddr6` converts the string representation in `IP` to a IPV6 host address and returns the host address.

Errors: On error, the address is filled with zeroes.

See also: `NetAddrToStr6` ([1070](#)), `HostAddrToStr6` ([1069](#)), `StrToHostAddr` ([1073](#))

33.4.38 StrToNetAddr

Synopsis: Convert a string to a network address.

Declaration: `function StrToNetAddr(IP: AnsiString) : in_addr`

Visibility: default

Description: `StrToNetAddr` converts the string representation in `IP` to a network address and returns the network address.

Errors: On error, the network address is filled with zeroes.

See also: `NetAddrToStr` ([1070](#)), `HostAddrToStr` ([1069](#)), `StrToHostAddr` ([1073](#))

33.4.39 StrToNetAddr6

Synopsis: Convert a string to a IPV6 network address

Declaration: `function StrToNetAddr6(IP: AnsiString) : Tin6_addr`

Visibility: default

Description: `StrToNetAddr6` converts the string representation in `IP` to a IPV6 network address and returns the network address.

Errors: On error, the address is filled with zeroes.

See also: `NetAddrToStr6` ([1070](#)), `HostAddrToStr6` ([1069](#)), `StrToHostAddr6` ([1073](#))

Chapter 34

Reference for unit 'strings'

34.1 Overview

This chapter describes the `STRINGS` unit for Free Pascal. This unit is system independent, and therefore works on all supported platforms.

34.2 Procedures and functions

34.2.1 `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` ([1084](#)), `StrPCopy` ([1085](#))

34.2.2 `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` ([1080](#))

Listing: `./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.

```

34.2.3 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` ([1081](#)).

Errors: None.

See also: `StrLComp` ([1081](#)), `StrIComp` ([1079](#)), `StrLComp` ([1082](#))

34.2.4 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` ([1085](#)), `StrLCopy` ([1081](#)), `StrECopy` ([1077](#))

Listing: ./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.

```

34.2.5 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 ([1084](#))

Listing: ./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.

```

34.2.6 strecopy

Synopsis: Copy a null-terminated string, return a pointer to the end.

Declaration: `function strcpy(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` ([1081](#)), `StrCopy` ([1076](#))

Listing: `./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.

34.2.7 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` ([1082](#))

Listing: `./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.

```

34.2.8 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` ([1081](#)), `StrComp` ([1076](#)), `StrLComp` ([1082](#))

Listing: `./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.

```

34.2.9 stripos

Synopsis: Return the position of a substring in a string, case insensitive.

Declaration: `function stripos(str1: pchar;str2: pchar) : pchar`

Visibility: default

Description: `stripos` 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: `striscan` ([1080](#)), `strpos` ([1086](#))

34.2.10 `striscan`

Synopsis: Scan a string for a character, case-insensitive

Declaration: `function striscan(p: pchar; c: Char) : pchar`

Visibility: default

Description: `striscan` does the same as `strscan` ([1087](#)) 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` ([1087](#)), `striscan` ([1086](#))

34.2.11 `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` ([1075](#))

Listing: `./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.

34.2.12 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` ([1076](#)), `StrIComp` ([1079](#)), `StrLComp` ([1082](#))

Listing: `./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.

34.2.13 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` ([1076](#)), `StrECopy` ([1077](#))

Listing: `./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.

```

34.2.14 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 ([1084](#))

Listing: ./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.

```

34.2.15 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` ([1079](#))

Errors: None.

See also: `StrLComp` ([1081](#)), `StrComp` ([1076](#)), `StrIComp` ([1079](#))

34.2.16 `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` ([1087](#))

Listing: `./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

`WriteLn ('Uppercase : ', StrUpper(P2));`

`StrLower (P1);`

`WriteLn ('Lowercase : ', P1);`

end.

34.2.17 `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` ([1081](#)), `StrCopy` ([1076](#))

Listing: ./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.

34.2.18 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 ([1076](#)), StrDispose ([1077](#))

Listing: ./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.

34.2.19 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 ([1085](#))

Listing: ./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.

34.2.20 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 ([1085](#))

Listing: ./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.

```

34.2.21 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 ([1087](#)), StrRScan ([1087](#))

Listing: ./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.

34.2.22 strriscan

Synopsis: Scan a string reversely for a character, case-insensitive

Declaration: `function strriscan(p: pchar;c: Char) : pchar`

Visibility: default

Description: `strriscan` does the same as `strrscan` ([1087](#)) 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` ([1087](#)), `striscan` ([1080](#))

34.2.23 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 (1087).

Errors: None.

See also: StrScan (1087), StrPos (1086)

34.2.24 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 (1087), StrPos (1086)

Listing: ./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.

34.2.25 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 (1083)

Errors: None.

See also: StrLower (1083)

Chapter 35

Reference for unit 'strutils'

35.1 Used units

Table 35.1: Used units by unit 'strutils'

Name	Page
System	1118
sysutils	1360

35.2 Constants, types and variables

35.2.1 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

35.2.2 Constants

`AnsiResemblesProc` : `TCompareTextProc` = `@SoundexProc`

This procedural variable is standard set to `SoundexProc` ([1113](#)) 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

```
StdSwitchChars = ['-','/']
```

Standard characters for the `SwitchChars` argument of `GetCmdLineArg` (1101).

```
StdWordDelims = [#0..' ', ',','.', ';','/','\',':', '','','"', ''] + Brackets
```

Standard word delimiter values.

```
WordDelimiters : Set of Char = [#0..#255] - ['a'..'z', 'A'..'Z', '1'..'9', '0']
```

Standard word delimiters, used in the `SearchBuf` (1111) call.

35.2.3 Types

```
TCompareTextProc = function(const AText: string;const AOther: string)
                    : Boolean
```

Function prototype for comparing two string in `AnsiResemblesText` (1094)

```
TRomanConversionStrictness = (rcsStrict, rcsRelaxed, rcsDontCare)
```

Table 35.2: 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.

```
TStringSeachOption = TStringSearchOption
```

There is an typo error in the original Borland StrUtils unit. This type just refers to the correct TStringSearchOption (1090) and is provided for compatibility only.

```
TStringSearchOption = (soDown, soMatchCase, soWholeWord)
```

Table 35.3: Enumeration values for type TStringSearchOption

Value	Explanation
soDown	Search in down direction.
soMatchCase	Match case
soWholeWord	Search whole words only.

Possible options for SearchBuf (1111) call.

```
TStringSearchOptions = Set of TStringSearchOption
```

Set of options for SearchBuf (1111) call.

35.3 Procedures and functions

35.3.1 AddChar

Synopsis: Add characters to the left of a string till a certain length

Declaration: `function AddChar(C: Char; 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 (1090), PadLeft (1106), PadRight (1107), PadCenter (1106)

35.3.2 AddCharR

Synopsis: Add chars at the end of a string till it reaches a certain length

Declaration: `function AddCharR(C: Char; 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 (1090)

35.3.3 AnsiContainsStr

Synopsis: Checks whether a string contains a given substring

Declaration: `function AnsiContainsStr(const AText: string;const ASubText: string)
: 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` (1091), `AnsiEndsStr` (1091), `AnsiIndexStr` (1092), `AnsiStartsStr` (1095)

35.3.4 AnsiContainsText

Synopsis: Check whether a string contains a certain substring, ignoring case.

Declaration: `function AnsiContainsText(const AText: string;const ASubText: string)
: 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-insensitive.

See also: `AnsiContainsStr` (1091), `AnsiEndsText` (1091), `AnsiIndexText` (1092), `AnsiStartsText` (1095)

35.3.5 AnsiEndsStr

Synopsis: Check whether a string ends with a certain substring

Declaration: `function AnsiEndsStr(const ASubText: string;const AText: string)
: 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` (1091), `AnsiStartsStr` (1095), `AnsiIndexStr` (1092), `AnsiContainsStr` (1091)

35.3.6 AnsiEndsText

Synopsis: Check whether a string ends with a certain substring, ignoring case.

Declaration: `function AnsiEndsText(const ASubText: string;const AText: string)
: 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-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 \(1095\)](#), [AnsiEndsStr \(1091\)](#), [AnsiIndexText \(1092\)](#), [AnsiContainsText \(1091\)](#)

35.3.7 AnsiIndexStr

Synopsis: Searches, observing case, for a string in an array of strings.

Declaration: `function AnsiIndexStr(const AText: string;
const AValues: Array of string) : 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 \(1092\)](#), [AnsiMatchStr \(1093\)](#), [AnsiMatchText \(1093\)](#)

35.3.8 AnsiIndexText

Synopsis: Searches, case insensitive, for a string in an array of strings.

Declaration: `function AnsiIndexText(const AText: string;
const AValues: Array of string) : 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 \(1092\)](#), [AnsiMatchStr \(1093\)](#), [AnsiMatchText \(1093\)](#)

35.3.9 AnsiLeftStr

Synopsis: Copies a number of characters starting at the left of a string

Declaration: `function AnsiLeftStr(const AText: AnsiString; const ACount: Integer)
: 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 \(1093\)](#), [AnsiRightStr \(1095\)](#), [LeftStr \(1104\)](#), [RightStr \(1110\)](#), [MidStr \(1105\)](#), [LeftBStr \(1104\)](#), [RightBStr \(1110\)](#), [MidBStr \(1105\)](#)

35.3.10 AnsiMatchStr

Synopsis: Check whether a string occurs in an array of strings, observing case.

Declaration: `function AnsiMatchStr(const AText: string;
const AValues: Array of string) : 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 observing case.

This function simply calls `AnsiIndexStr` (1092) and checks whether it returns -1 or not.

35.3.11 AnsiMatchText

Synopsis: Check whether a string occurs in an array of strings, disregarding case.

Declaration: `function AnsiMatchText(const AText: string;
const AValues: Array of string) : 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` (1092) and checks whether it returns -1 or not.

35.3.12 AnsiMidStr

Synopsis: Returns a number of characters copied from a given location in a string

Declaration: `function AnsiMidStr(const AText: AnsiString; const AStart: Integer;
const ACount: Integer) : 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` (1092), `AnsiRightStr` (1095), `LeftStr` (1104), `RightStr` (1110), `MidStr` (1105), `LeftBStr` (1104), `RightBStr` (1110), `MidBStr` (1105)

35.3.13 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` (1089) can be used for this.

35.3.14 AnsiReplaceStr

Synopsis: Search and replace all occurrences of a string, case sensitive.

Declaration: `function AnsiReplaceStr(const AText: string;const AFromText: string;
const AToText: string) : string`

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` ([1094](#)), `SearchBuf` ([1111](#))

35.3.15 AnsiReplaceText

Synopsis: Search and replace all occurrences of a string, case insensitive.

Declaration: `function AnsiReplaceText(const AText: string;const AFromText: string;
const AToText: string) : string`

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` ([1094](#)), `SearchBuf` ([1111](#))

35.3.16 AnsiResemblesText

Synopsis: Check whether 2 strings resemble each other.

Declaration: `function AnsiResemblesText(const AText: string;const AOther: string)
: Boolean`

Visibility: default

Description: `AnsiResemblesText` will check whether `AnsiResemblesProc` ([1088](#)) is set. If it is not set, `False` is returned. If it is set, `AText` and `AOtherText` are passed to it and its result is returned.

Errors: None.

See also: `AnsiResemblesProc` ([1088](#)), `SoundexProc` ([1113](#))

35.3.17 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 an anagram.

Errors: None.

35.3.18 AnsiRightStr

Synopsis: Copies a number of characters starting at the right of a string

Declaration: `function AnsiRightStr(const AText: AnsiString; const ACount: Integer)
: 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` (1092), `AnsiMidStr` (1093), `LeftStr` (1104), `RightStr` (1110), `MidStr` (1105), `LeftBStr` (1104), `RightBStr` (1110), `MidBStr` (1105)

35.3.19 AnsiStartsStr

Synopsis: Check whether a string starts with a given substring, observing case

Declaration: `function AnsiStartsStr(const ASubText: string; const AText: string)
: 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` (1091), `AnsiStartsStr` (1095), `AnsiIndexStr` (1092), `AnsiContainsStr` (1091)

35.3.20 AnsiStartsText

Synopsis: Check whether a string starts with a given substring, ignoring case

Declaration: `function AnsiStartsText(const ASubText: string; const AText: string)
: 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` (1091), `AnsiStartsStr` (1095), `AnsiIndexText` (1092), `AnsiContainsText` (1091)

35.3.21 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` ([1102](#))

35.3.22 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` ([1096](#)) with the space (ASCII code 32) as the symbol argument.

Errors: None.

See also: `Copy2Symb` ([1096](#)), `Copy2SpaceDel` ([1096](#))

35.3.23 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` ([1097](#)) with the space (ASCII code 32) as the symbol argument.

Errors: None.

See also: `Copy2SymbDel` ([1097](#)), `Copy2Space` ([1096](#))

35.3.24 Copy2Symb

Synopsis: Returns all characters in a string till a given character (not included).

Declaration: `function Copy2Symb(const S: string; Symb: Char) : 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 \(1096\)](#), [Copy2SymbDel \(1097\)](#)

35.3.25 Copy2SymbDel

Synopsis: Deletes and returns all characters in a string till a given character (not included).

Declaration: `function Copy2SymbDel (var S: string; Symb: Char) : 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 \(1096\)](#), [Copy2Symb \(1096\)](#)

35.3.26 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 \(1101\)](#), [IntToBin \(1102\)](#), [intToRoman \(1103\)](#), [RomanToInt \(1110\)](#)

35.3.27 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 \(1113\)](#) function. The result is the soundex string corresponding to `AValue`.

Errors: None.

See also: [SoundexInt \(1113\)](#), [DecodeSoundexWord \(1098\)](#), [Soundex \(1112\)](#)

35.3.28 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` (1114) function. The result is the soundex string corresponding to `AValue`.

Errors: None.

See also: `SoundexInt` (1113), `DecodeSoundexInt` (1097), `Soundex` (1112)

35.3.29 DelChars

Synopsis: Delete all occurrences of a given character from a string.

Declaration: `function DelChars(const S: string; Chr: Char) : string`

Visibility: default

Description: `DelChars` returns a copy of `S` with all `Chr` characters removed from it.

Errors: None.

See also: `DelSpace` (1098), `DelSpace1` (1098)

35.3.30 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` (1098), `DelSpace1` (1098)

35.3.31 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` (1098), `DelSpace` (1098)

35.3.32 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'
```

Errors: None.

35.3.33 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` (1100), an empty string is a valid return value, i.e. a part can be empty.

The pre-defined constant `StdWordDelims` (1089) can be used for the `Delims` argument. The pre-defined constant `Brackets` (1088) would be better suited the `Delims` argument e.g. in case factors in a mathematical expression are searched.

Errors: None.

See also: `ExtractSubStr` (1099), `ExtractWord` (1100), `ExtractWordPos` (1100)

35.3.34 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 pre-defined constant `StdWordDelims` (1089) can be used for the `Delims` argument.

Errors: None.

See also: `ExtractDelimited` (1099), `ExtractWord` (1100), `ExtractWordPos` (1100)

35.3.35 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` (1099), 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 pre-defined constant `StdWordDelims` (1089) can be used for the `WordDelims` argument.

Errors: None.

See also: `ExtractWordPos` (1100), `ExtractSubStr` (1099), `ExtractDelimited` (1099), `IsWordPresent` (1104), `WordCount` (1116), `WordPosition` (1116)

35.3.36 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; var 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` (1099), 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 pre-defined constant `StdWordDelims` (1089) can be used for the `WordDelims` argument.

Errors: None.

See also: `ExtractWord` (1100), `ExtractSubStr` (1099), `IsWordPresent` (1104), `WordCount` (1116), `WordPosition` (1116)

35.3.37 FindPart

Synopsis: Search for a substring in a string, using wildcards.

Declaration: `function FindPart(const HelpWilds: string; const InputStr: string)
: Integer`

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` ([1111](#))

35.3.38 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` ([1089](#)) 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` ([1089](#))

35.3.39 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` ([1097](#)), `IntToBin` ([1102](#)), `intToRoman` ([1103](#)), `RomanToInt` ([1110](#))

35.3.40 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` ([1095](#))

35.3.41 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`

Visibility: default

Description: `IfThen` returns `ATrue` if `AValue` is `True`, and returns `AFalse` if `AValue` is `false`.

Errors: None.

See also: `AnsiMatchStr` ([1093](#)), `AnsiMatchText` ([1093](#))

35.3.42 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 its binary (base 2) representation. The resulting string contains at least `Digits` digits, with spaces inserted every `Spaces` digits. `Spaces` should be a nonzero value. If `Digits` is larger than 32, it is truncated to 32.

Errors: If `spaces` is zero, a division by zero error will occur.

See also: `Hex2Dec` ([1101](#)), `IntToRoman` ([1103](#))

35.3.43 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` ([1110](#)), `Hex2Dec` ([1101](#)), `IntToBin` ([1102](#))

35.3.44 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` ([1103](#)), `FindPart` ([1100](#)), `IsWordPresent` ([1104](#))

35.3.45 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` ([1111](#)), `FindPart` ([1100](#))

35.3.46 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` (1111) function with the `soWholeWords` option specified.

Errors: None.

See also: `SearchBuf` (1111)

35.3.47 LeftBStr

Synopsis: Copies Count characters starting at the left of a string.

Declaration: `function LeftBStr(const AText: AnsiString;const AByteCount: Integer)
: 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` (1104), `AnsiLeftStr` (1092), `RightBStr` (1110), `MidBStr` (1105)

35.3.48 LeftStr

Synopsis: Copies Count characters starting at the left of a string.

Declaration: `function LeftStr(const AText: AnsiString;const ACount: Integer)
: AnsiString
function LeftStr(const AText: WideString;const ACount: Integer)
: 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` (1104), `AnsiLeftStr` (1092), `RightStr` (1110), `MidStr` (1105)

35.3.49 MidBStr

Synopsis: Copies a number of characters starting at a given position in a string.

Declaration: `function MidBStr(const AText: AnsiString; const AByteStart: Integer;
const AByteCount: Integer) : 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 \(1104\)](#), [AnsiMidStr \(1093\)](#), [RightBStr \(1110\)](#), [MidStr \(1105\)](#)

35.3.50 MidStr

Synopsis: Copies a number of characters starting at a given position in a string.

Declaration: `function MidStr(const AText: AnsiString; const AStart: Integer;
const ACount: Integer) : AnsiString`
`function MidStr(const AText: WideString; const AStart: Integer;
const ACount: Integer) : 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 \(1104\)](#), [AnsiMidStr \(1093\)](#), [RightStr \(1110\)](#), [MidBStr \(1105\)](#)

35.3.51 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) : Integer`

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 \(1116\)](#), [FindPart \(1100\)](#)

35.3.52 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` ([1101](#)), `Numb2USA` ([1106](#))

35.3.53 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.

35.3.54 PadCenter

Synopsis: Pad the string to a certain length, so the string is centered.

Declaration: `function PadCenter(const S: string;Len: Integer) : 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 largert than `Len`, no spaces are added, and the string `S` is returned as-is.

Errors: None.

See also: `PadLeft` ([1106](#)), `PadRight` ([1107](#)), `AddChar` ([1090](#)), `AddCharR` ([1090](#))

35.3.55 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 largert 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` ([1106](#)), `PadCenter` ([1106](#)), `AddChar` ([1090](#)), `AddCharR` ([1090](#))

35.3.56 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 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`, left-justified on length `Len`.

Errors: None.

See also: `PadLeft` ([1106](#)), `PadCenter` ([1106](#)), `AddChar` ([1090](#)), `AddCharR` ([1090](#))

35.3.57 PosEx

Synopsis: Search for the occurrence of a character in a string, starting at a certain position.

Declaration: `function PosEx(const SubStr: string; const S: string; Offset: Cardinal) : Integer`
`function PosEx(const SubStr: string; const S: string) : Integer`
`function PosEx(c: Char; const S: string; Offset: Cardinal) : Integer`

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` ([1105](#)), `AnsiContainsText` ([1091](#)), `AnsiContainsStr` ([1091](#))

35.3.58 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) : Integer`
`function PosSet(const c: string; const s: ansistring) : Integer`

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` ([1107](#)), `PosSetEx` ([1107](#)), `#rtl.system.pos` ([1287](#)), `RPosEx` ([1111](#))

35.3.59 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) : Integer`
`function PosSetEx(const c: string; const s: ansistring; count: Integer) : Integer`

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` (1107), `PosSet` (1107), `#rtl.system.pos` (1287), `RPosEx` (1111)

35.3.60 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` (1093), `AnsiMatchText` (1093)

35.3.61 Removeleadingchars

Synopsis: Remove any leading characters in a set from a string

Declaration: `procedure Removeleadingchars(var S: AnsiString; 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` (1510) which used whitespace as the set.

Errors: None.

See also: `TrimLeft` (1510), `RemoveTrailingChars` (1109), `RemovePadChars` (1108), `TrimLeftSet` (1115)

35.3.62 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` (1509) which used whitespace as the set.

Errors: None.

See also: `Trim` (1509), `RemoveLeadingChars` (1108), `RemoveTrailingChars` (1109), `TrimSet` (1115), `TrimLeftSet` (1115), `TrimRightSet` (1115)

35.3.63 RemoveTrailingChars

Synopsis: Remove any trailing characters in a set from a string

Declaration: `procedure RemoveTrailingChars(var S: AnsiString; 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` (1510) which used whitespace as the set.

See also: `TrimRight` (1510), `RemoveLeadingChars` (1108), `TrimRightSet` (1115)

35.3.64 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` (1487) with the appropriate options.

See also: `StringReplace` (1487), `ReplaceText` (1109)

35.3.65 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` (1487) with the appropriate options.

See also: `StringReplace` (1487), `ReplaceText` (1109)

35.3.66 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` (1108)

35.3.67 RightBStr

Synopsis: Copy a given number of characters (bytes), counting from the right of a string.

Declaration: `function RightBStr(const AText: AnsiString; const AByteCount: Integer)
: 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` (1104), `AnsiRightStr` (1095), `RightStr` (1110), `MidBStr` (1105)

35.3.68 RightStr

Synopsis: Copy a given number of characters, counting from the right of a string.

Declaration: `function RightStr(const AText: AnsiString; const ACount: Integer)
: AnsiString`
`function RightStr(const AText: WideString; const ACount: Integer)
: 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` (1104), `AnsiRightStr` (1095), `RightBStr` (1110), `MidStr` (1105)

35.3.69 RomanToInt

Synopsis: Convert a string with a Roman number to its 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` (1089).

Errors: On error, a `EConvertError` (1088) exception is raised.

See also: `TRomanConversionStrictness` (1089), `IntToRoman` (1103), `Hex2Dec` (1101), `Numb2Dec` (1106)

35.3.70 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` (1089), `TryRomanToInt` (1116), `RomanToInt` (1110), `IntToRoman` (1103)

35.3.71 RPos

Synopsis: Find last occurrence of substring or character in a string

Declaration: `function RPos(c: Char;const S: AnsiString) : Integer; Overload
function RPos(const Substr: AnsiString;const Source: AnsiString)
: Integer; 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.

See also: `RPosEx` (1111)

35.3.72 RPosEx

Synopsis: Find last occurrence substring or character in a string, starting at a certain position

Declaration: `function RPosEX(C: Char;const S: AnsiString;offs: Cardinal) : Integer
; Overload
function RPosEx(const Substr: AnsiString;const Source: AnsiString;
offs: Cardinal) : Integer; 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.

See also: `RPos` (1111)

35.3.73 SearchBuf

Synopsis: Search a buffer for a certain string.

Declaration: `function SearchBuf(Buf: PChar;BufLen: Integer;SelStart: Integer;
SelLength: Integer;SearchString: string;
Options: TStringSearchOptions) : PChar
function SearchBuf(Buf: PChar;BufLen: Integer;SelStart: Integer;
SelLength: Integer;SearchString: string) : PChar`

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 `SelStart+SelLength`. If a match is found, a pointer to the position of the match is returned. The parameter `Options` (1090) specifies how the search is conducted. It is a set of the following options:

Table 35.4:

Option	Effect
<code>soDown</code>	Searches forward, starting at the end of the selection. Default is searching up
<code>soMatchCase</code>	Observe case when searching. Default is to ignore case.
<code>soWholeWord</code>	Match only whole words. Default also returns parts of words

The standard constant `WordDelimiters` (1089) 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` (1100), `ExtractWord` (1100), `ExtractWordPos` (1100), `ExtractSubStr` (1099), `IsWordPresent` (1104)

35.3.74 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` (1112), `SoundexInt` (1113), `SoundexProc` (1113), `SoundexWord` (1114), `SoundexSimilar` (1113)

35.3.75 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 \(1112\)](#), [SoundexInt \(1113\)](#), [SoundexProc \(1113\)](#), [SoundexWord \(1114\)](#), [SoundexSimilar \(1113\)](#)

35.3.76 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 \(1112\)](#) code (with length `ALength`, default 4) of `AText`, and converts the code to an integer value.

Errors: None.

See also: [Soundex \(1112\)](#), [SoundexCompare \(1112\)](#), [SoundexProc \(1113\)](#), [SoundexWord \(1114\)](#), [SoundexSimilar \(1113\)](#)

35.3.77 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 \(1094\)](#) 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 \(1112\)](#), [SoundexCompare \(1112\)](#), [SoundexInt \(1113\)](#), [SoundexWord \(1114\)](#), [SoundexSimilar \(1113\)](#)

35.3.78 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 \(1112\)](#), [SoundexCompare \(1112\)](#), [SoundexInt \(1113\)](#), [SoundexProc \(1113\)](#), [SoundexWord \(1114\)](#), [Soundex \(1112\)](#)

35.3.79 SoundexWord

Synopsis: Calculate a word-sized soundex value

Declaration: `function SoundexWord(const AText: string) : Word`

Visibility: default

Description: `SoundexInt` computes the Soundex (1112) 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 (1112), SoundexCompare (1112), SoundexInt (1113), SoundexProc (1113), SoundexSimilar (1113)

35.3.80 StringsReplace

Synopsis: Replace occurrences of a set of strings to another set of strings

Declaration: `function StringsReplace(const S: string; OldPattern: Array of string;
NewPattern: Array of string; 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` (1487).

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` (1487), `TReplaceFlags` (1380)

35.3.81 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` (1100), `DelChars` (1098), `DelSpace` (1098), `ExtractSubStr` (1099), `DupeString` (1099)

35.3.82 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` (1114), `FindPart` (1100), `ExtractWord` (1100), `DelChars` (1098), `DelSpace` (1098), `DelSpace1` (1098), `DupeString` (1099)

35.3.83 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` (1108), but returns the resulting string.

Errors: None.

See also: `TrimLeft` (1510), `RemoveLeadingChars` (1108), `RemoveTrailingChars` (1109), `RemovePadChars` (1108), `TrimSet` (1115), `TrimRightSet` (1115)

35.3.84 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: `TrimRightSet` performs the same action as `RemoveTrailingChars` (1109), but returns the resulting string.

Errors: None.

See also: `TrimRight` (1510), `RemoveLeadingChars` (1108), `RemoveTrailingChars` (1109), `RemovePadChars` (1108), `TrimSet` (1115), `TrimLeftSet` (1115)

35.3.85 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` (1108), but returns the resulting string.

Errors: None.

See also: `Trim` (1509), `RemoveLeadingChars` (1108), `RemoveTrailingChars` (1109), `RemovePadChars` (1108), `TrimRightSet` (1115), `TrimLeftSet` (1115)

35.3.86 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` (1089), `RomanToIntDef` (1111), `RomanToInt` (1110), `IntToRoman` (1103)

35.3.87 WordCount

Synopsis: Count the number of words in a string.

Declaration: `function WordCount (const S: string; const WordDelims: TSysCharSet)
: Integer`

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 pre-defined `StdWordDelims` (1089) constant can be used for the `WordDelims` argument.

Errors: None.

See also: `WordPosition` (1116), `StdWordDelims` (1089), `ExtractWord` (1100), `ExtractWordPos` (1100)

35.3.88 WordPosition

Synopsis: Search position of Nth word in a string.

Declaration: `function WordPosition (const N: Integer; const S: string;
const WordDelims: TSysCharSet) : Integer`

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 pre-defined `StdWordDelims` (1089) constant can be used for the `WordDelims` argument.

Errors: None

See also: `WordCount` (1116), `StdWordDelims` (1089), `ExtractWord` (1100), `ExtractWordPos` (1100)

35.3.89 XorDecode

Synopsis: Decode a string encoded with XorEncode (1117)

Declaration: `function XorDecode(const Key: string;const Source: string) : string`

Visibility: default

Description: `XorDecode` decodes `Source` and returns the original string that was encrypted using `XorEncode` (1117) 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` (1117), `XorString` (1117)

35.3.90 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: string;const Source: string) : string`

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` (1116) 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` (1116), `XorString` (1117), `Hex2Dec` (1101)

35.3.91 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` (1117) function instead, which will representing 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` (1117), `XorDecode` (1116)

Chapter 36

Reference for unit 'System'

36.1 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 pre-defined 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.

36.2 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 36.1:

Name	Description
DefaultSystemCodePage (1181)	Actual code page to use when CP_ACP (1124) is encountered
DefaultUnicodeCodePage (1181)	Code page for new unicode strings
DefaultFileSystemCodePage (1180)	Codepage to use when sending strings to single-byte OS filesystem routines.
DefaultRTLFileSystemCodePage (1180)	Codepage to use when receiving strings from single-byte OS filesystem routines.

The windows code page identifiers are used. There are 3 special codepage identifiers:

Table 36.2:

Name	Description
CP_ACP (1124)	Currently set default systsem cpdepage
CP_OEMCP (1124)	OEM (console) code page (only on windows)
CP_NONE (1124)	Indicates absence of code page information for a string
DefaultRTLFileSystemCodePage (1180)	

The following routines may perform code page conversions:

Table 36.3:

Name	Description
LowerCase (1254)	Return lowercase version of a string.
UpCase (1333)	Convert a string to all uppercase.
GetDir (1230)	Return the current directory
MkDir (1255)	Create a new directory.
ChDir (1197)	Change current working directory.
RmDir (1297)	Remove directory when empty.
Assign (1188)	Assign a name to a file
Erase (1217)	Delete a file from disk
Rename (1295)	Rename file on disk
Read (1291)	Read from a text file into variable
ReadLn (1292)	Read from a text file into variable and goto next line
Write (1340)	Write variable to a text file
WriteLn (1340)	Write variable to a text file and append newline
ReadStr (1292)	Read variables from a string
WriteStr (1341)	Write variables to a string
Insert (1244)	Insert one string in another.
Copy (1205)	Copy part of a string.
Delete (1208)	Delete part of a string.
SetString (1311)	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 fpwdestring or cwstring units must be used.

36.3 Miscellaneous functions

Functions that do not belong in one of the other categories.

Table 36.4:

Name	Description
Assert (1188)	Conditionally abort program with error
Break (1193)	Abort current loop
Continue (1204)	Next cycle in current loop
Exclude (1217)	Exclude an element from a set
Exit (1219)	Exit current function or procedure
Include (1239)	Include an element into a set
LongJump (1253)	Jump to execution point
Ord (1284)	Return ordinal value of enumerated type
Pred (1288)	Return previous value of ordinal type
SetJump (1308)	Mark execution point for jump
SizeOf (1315)	Return size of variable or type
Succ (1322)	Return next value of ordinal type

36.4 Operating System functions

Functions that are connected to the operating system.

Table 36.5:

Name	Description
Chdir (1197)	Change working directory
Getdir (1230)	Return current working directory
Halt (1235)	Halt program execution
Paramcount (1285)	Number of parameters with which program was called
Paramstr (1285)	Retrieve parameters with which program was called
Mkdir (1255)	Make a directory
Rmdir (1297)	Remove a directory
Runerror (1302)	Abort program execution with error condition

36.5 String handling

All things connected to string handling.

Table 36.6:

Name	Description
BinStr (1191)	Construct binary representation of integer
Chr (1197)	Convert ASCII code to character
Concat (1203)	Concatenate two strings
Copy (1205)	Copy part of a string
Delete (1208)	Delete part of a string
HexStr (1236)	Construct hexadecimal representation of integer
Insert (1244)	Insert one string in another
Length (1250)	Return length of string
Lowercase (1254)	Convert string to all-lowercase
OctStr (1257)	Construct octal representation of integer
Pos (1287)	Calculate position of one string in another
SetLength (1309)	Set length of a string
SetString (1311)	Set contents and length of a string
Str (1319)	Convert number to string representation
StringOfChar (1320)	Create string consisting of a number of characters
Uppcase (1333)	Convert string to all-uppercase
Val (1336)	Convert string to number

36.6 Mathematical routines

Functions connected to calculating and converting numbers.

Table 36.7:

Name	Description
Abs (1184)	Calculate absolute value
Arctan (1187)	Calculate inverse tangent
Cos (1206)	Calculate cosine of angle
Dec (1207)	Decrease value of variable
Exp (1220)	Exponentiate
Frac (1228)	Return fractional part of floating point value
Hi (1236)	Return high byte/word of value
Inc (1238)	Increase value of variable
Int (1245)	Calculate integer part of floating point value
Ln (1251)	Calculate logarithm
Lo (1252)	Return low byte/word of value
Odd (1258)	Is a value odd or even ?
Pi (1286)	Return the value of pi
Random (1290)	Generate random number
Randomize (1290)	Initialize random number generator
Round (1300)	Round floating point value to nearest integer number
Sin (1315)	Calculate sine of angle
Sqr (1317)	Calculate the square of a value
Sqrt (1318)	Calculate the square root of a value
Swap (1323)	Swap high and low bytes/words of a variable
Trunc (1329)	Truncate a floating point value

36.7 Memory management functions

Functions concerning memory issues.

Table 36.8:

Name	Description
Addr (1185)	Return address of variable
Assigned (1189)	Check if a pointer is valid
CompareByte (1198)	Compare 2 memory buffers byte per byte
CompareChar (1199)	Compare 2 memory buffers byte per byte
CompareDWord (1201)	Compare 2 memory buffers byte per byte
CompareWord (1202)	Compare 2 memory buffers byte per byte
CSeg (1206)	Return code segment
Dispose (1209)	Free dynamically allocated memory
DSeg (1210)	Return data segment
FillByte (1222)	Fill memory region with 8-bit pattern
FillChar (1222)	Fill memory region with certain character
FillDWord (1223)	Fill memory region with 32-bit pattern
FillQWord (1224)	Fill memory region with 64-bit pattern
FillWord (1224)	Fill memory region with 16-bit pattern
Freemem (1228)	Release allocated memory
Getmem (1231)	Allocate new memory
GetMemoryManager (1231)	Return current memory manager
High (1237)	Return highest index of open array or enumerated
IndexByte (1240)	Find byte-sized value in a memory range
IndexChar (1240)	Find char-sized value in a memory range
IndexDWord (1241)	Find DWord-sized (32-bit) value in a memory range
IndexQWord (1242)	Find QWord-sized value in a memory range
IndexWord (1242)	Find word-sized value in a memory range
IsMemoryManagerSet (1249)	Is the memory manager set
Low (1253)	Return lowest index of open array or enumerated
Move (1255)	Move data from one location in memory to another
MoveChar0 (1256)	Move data till first zero character
New (1256)	Dynamically allocate memory for variable
Ofs (1258)	Return offset of variable
Ptr (1289)	Combine segment and offset to pointer
ReAllocMem (1294)	Resize a memory block on the heap
Seg (1306)	Return segment
SetMemoryManager (1309)	Set a memory manager
Sptr (1317)	Return current stack pointer
SSeg (1318)	Return stack segment register value

36.8 File handling functions

Functions concerning input and output from and to file.

Table 36.9:

Name	Description
Append (1186)	Open a file in append mode
Assign (1188)	Assign a name to a file
Blockread (1192)	Read data from a file into memory
Blockwrite (1193)	Write data from memory to a file
Close (1198)	Close a file
Eof (1215)	Check for end of file
Eoln (1216)	Check for end of line
Erase (1217)	Delete file from disk
Filepos (1220)	Position in file
Filesize (1221)	Size of file
Flush (1226)	Write file buffers to disk
IOresult (1247)	Return result of last file IO operation
Read (1291)	Read from file into variable
Readln (1292)	Read from file into variable and goto next line
Rename (1295)	Rename file on disk
Reset (1295)	Open file for reading
Rewrite (1296)	Open file for writing
Seek (1304)	Set file position
SeekEof (1304)	Set file position to end of file
SeekEoln (1305)	Set file position to end of line
SetTextBuf (1311)	Set size of file buffer
Truncate (1329)	Truncate the file at position
Write (1340)	Write variable to file
WriteLn (1340)	Write variable to file and append newline

36.9 Run-Time Error behaviour

The sytem 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 (1360) 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.

36.10 Constants, types and variables

36.10.1 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 (1137).

```
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 AbstractErrorProc constant is used when an assert error occurs. If it is not set, then the standard error handling is done: The assertion error message is printed, together with the location of the assertion, and A stack dump is performed, and 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.

```
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`

`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` (1172) 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 driveletters, 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` ([1360](#)) 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 = ($80004002)
```

Interface call result: Error: not an interface

```
E_NOTIMPL = ($80004001)
```

Interface call result: Interface not implemented

`E_UNEXPECTED = ($8000FFFF)`

Interface call result: Unexpected error

`Filemode : Byte = 2`

Default file mode for untyped files.

`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 filesystem: 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 filesystem: 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_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_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_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_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_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_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_new_x = 46

FPC compiler internal procedure index: new

fpc_in_ofs_x = 21

FPC compiler internal procedure index: ofs

fpc_in_ord_x = 5

FPC compiler internal procedure index: ord

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_x = 25

FPC compiler internal procedure index: reset

fpc_in_rewrite_typedfile = 33

FPC compiler internal procedure index: rewrite

fpc_in_rewrite_x = 26

FPC compiler internal procedure index: rewrite

fpc_in_rol_x = 67

fpc_in_rol_x_x = 68

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_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_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_objc_encode_x = 71

FPC compiler internal procedure index: encode

fpc_objc_protocol_x = 70

FPC compiler internal procedure index: protocol

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

`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` ([1235](#)) 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 = (ValSInt)`

Maximum String-size value.

`maxSmallint = 32767`

Maximum smallint value.

`MaxUIntValue = (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`).

`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.

`RaiseMaxFrameCount : LongInt = 16`

Maximum number of frames to include in `TExceptObject` ([1163](#))

`RaiseProc : TExceptProc = Nil`

Procedure to raise an exception.

`RT_ACCELERATOR = (9)`

Constant identifying an accelerator resource

`RT_ANICURSOR = (21)`

This constant can be used to specify a resource of type "animated cursor".

`RT_ANIICON = (22)`

This constant can be used to specify a resource of type "animated icon".

`RT_BITMAP = (2)`

Constant identifying a bitmap resource

`RT_CURSOR = (1)`

Constant identifying a cursor resource

`RT_DIALOG = (5)`

Constant identifying a dialog resource

`RT_FONT = (8)`

Constant identifying a font resource

`RT_FONTDIR = (7)`

Constant identifying a font directory resource

`RT_GROUP_CURSOR = (12)`

Constant identifying a group cursor resource

`RT_GROUP_ICON = (14)`

Constant identifying a group icon resource

`RT_HTML = (23)`

This constant can be used to specify a resource of type "HTML data".

`RT_ICON = (3)`

Constant identifying an icon resource

`RT_MANIFEST = (24)`

This constant can be used to specify a resource of type "Manifest".

`RT_MENU = (4)`

Constant identifying a menu resource

`RT_MESSAGETABLE = (11)`

Constant identifying a message data resource

`RT_RCDATA = (10)`

Constant identifying a binary data resource

`RT_STRING = (6)`

Constant identifying a string table resource

`RT_VERSION = (16)`

Constant identifying a version info resource

`RuntimeErrorExitCodes : Array[TRuntimeError] of Byte = (0, 203, 204, 200, 201, 215, ...)`

This array is used by the `Error` (1217) routine to convert a `TRuntimeError` (1170) 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.

`sLineBreak = LineEnding`

`sLineBreak` is an alias for `LineEnding` (1137) 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

`TextRecBufSize = 256`

`TextRecBufSize` is the default buffer size for text files. The actual buffer can be set to another size using `SetTextBuf` ([1311](#)).

`TextRecNameLength = 256`

`TextRecNameLength` is the maximum filename size for text files.

`ThreadingAlreadyUsed : Boolean = False`

Internal constant for the threading system. Don't use.

`UnixGetModuleByAddrHook : procedure(addr: pointer; var baseaddr: pointer; var filename`

`UnixGetModuleByAddrHook` is used on unix systems to retrieve a module name based on an address. It is used in the `exeinfo` ([581](#)) 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

`vareempty = 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 ([1257](#)) variant

`varolestr = 8`

Variant type: OLE string (widestring)

`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 `tvardata` (1175).

```
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 `tvardata` (1175).

```
varvariant = 12
```

Variant type: Variant (arrays only)

```
varword = 18
```

Variant type: Word (16 bit)

```
varword64 = varqword
```

Variant type: Word (64-bit)

```
vmtAfterConstruction = vmtMethodStart + ((codepointer) * 5)
```

VMt Layout: ?

```
vmtAutoTable = vmtParent + ((pointer) * 7)
```

VMt layout: ?

```
vmtBeforeDestruction = vmtMethodStart + ((codepointer) * 6)
```

VMt Layout: ?

```
vmtClassName = vmtParent + (pointer)
```

VMt Layout: location of class name.

`vmtDefaultHandler = vmtMethodStart + ((codepointer) * 4)`

VM T Layout: ?

`vmtDefaultHandlerStr = vmtMethodStart + ((codepointer) * 7)`

VM T Layout: ?

`vmtDestroy = vmtMethodStart`

VM T Layout: Location of destructor pointer.

`vmtDispatch = vmtMethodStart + ((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` ([1356](#))

`vmtDispatchStr = vmtMethodStart + ((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` ([1356](#))

`vmtDynamicTable = vmtParent + ((pointer) * 2)`

VM T Layout: location of dynamic methods table.

`vmtEquals = vmtMethodStart + ((codepointer) * 10)`

`vmtEquals` contains the offset from the VMT start, of the location of the `TObject.Equals` ([1359](#)) method pointer.

`vmtFieldTable = vmtParent + ((pointer) * 4)`

VM T Layout: Location of fields table.

`vmtFreeInstance = vmtMethodStart + ((codepointer) * 2)`

VM T Layout: location of `FreeInstance` method.

`vmtGetHashCode = vmtMethodStart + ((codepointer) * 11)`

`vmtGetHashCode` contains the offset from the VMT start, of the location of the `TObject.GetHashCode` ([1359](#)) method pointer.

`vmtInitTable = vmtParent + ((pointer) * 6)`

VM T Layout: ?

`vmtInstanceSize = 0`

VM T Layout: Location of class instance size in VMT

```
vmtIntfTable = vmtParent + ((pointer) * 8)
```

VMT layout: Interface table

```
vmtMethodStart = vmtParent + ((pointer) * 10)
```

VMT layout: start of method table.

```
vmtMethodTable = vmtParent + ((pointer) * 3)
```

VMT Layout: Method table start.

```
vmtMsgStrPtr = vmtParent + ((pointer) * 9)
```

VMT layout: message strings table.

```
vmtNewInstance = vmtMethodStart + (codepointer)
```

VMT Layout: location of NewInstance method.

```
vmtParent = (SizeInt) * 2
```

VMT Layout: location of pointer to parent VMT.

```
vmtSafeCallException = vmtMethodStart + ((codepointer) * 3)
```

VMT Layout: ?

```
vmtToString = vmtMethodStart + ((codepointer) * 12)
```

vmtToString contains the offset from the VMT start, of the location of the TObjct.ToString (1359) method pointer.

```
vmtTypeInfo = vmtParent + ((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

36.10.2 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.

`CodePtrInt = PtrInt`

CodePtrInt is a signed integer with the same size as CodePointer (1148)

`CodePtrUInt = PtrUInt`

CodePtrUInt is an unsigned integer with the same size as CodePointer (1148)

`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 (1214) call. It is called for all languages for a resource of the specified type and name, and is passed the ModuleHandle, ResourceType, 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
```

EnumResNameProcs used in the EnumResourceNames (1215) 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 (1215) 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
  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.

`IntegerArray = Array[0..$effffff] of Integer`

Generic array of integer.

`IntPtr = PtrInt`

A signed integer with the same size in bytes as pointer

```
jmp_buf = packed record
    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 (1151) type.

`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.

`NativeUInt = PtrUInt`

`NativeInt` 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.

`PAnsiChar = PChar`

Alias for PChar (1151) type.

`PAnsiString = ^AnsiString`

Pointer to an ansistring type.

`PBoolean = ^Boolean`

Pointer to a Boolean type.

`PByte = ^Byte`

Pointer to byte (1148) type

`pcallldesc = ^tcallldesc`

Pointer to TCallDesc (1161) record.

`PCardinal = ^Cardinal`

Pointer to Cardinal (1148) 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 (1161)

`PCodePointer = ^CodePointer`

PCodePointer is a typed pointer to CodePointer (1148).

PComp = ^Comp

PComp is a pointer to a complex type.

PCurrency = ^Currency

Pointer to currency type.

PDate = ^TDateTime

Pointer to a TDateTime (1161) type.

PDateTime = ^TDateTime

Pointer to TDateTime

PDispatch = ^IDispatch

Pointer to IDispatch (1342) interface type

pdispdesc = ^tdispdesc

Pointer to tdispdesc (1161) record

PDouble = ^Double

Pointer to double-sized float value.

PDWord = ^DWord

Pointer to DWord (1148) type

pdynarrayindex = ^tdynarrayindex

Pointer to tdynarrayindex (1162) type.

pdynarraytypeinfo = ^tdynarraytypeinfo

Pointer to TDynArrayTypeInfo (1162) type.

PError = ^TError

Pointer to an Error (1217) type.

PEventState = pointer

Pointer to EventState, which is an opaque type.

PExceptObject = ^TExceptObject

Pointer to Exception handler procedural type TExceptProc ([1163](#))

PExtended = ^Extended

Pointer to extended-sized float value.

PFileTextRecChar = ^TFileTextRecChar

PFileTextRecChar is a typed pointer to TFileTextRecChar ([1164](#)).

PGuid = ^TGuid

Pointer to TGUID ([1166](#)) type.

pInt16 = PSmallInt

Pointer to Int16 ([1150](#)) type

pInt32 = PLongint

Pointer to Int32 ([1150](#)) type

PInt64 = ^Int64

Pointer to Int64 type

pInt8 = PShortInt

Pointer to Int8 ([1150](#)) type

PInteger = ^Integer

Pointer to integer ([1150](#)) type

PIntegerArray = ^IntegerArray

Pointer to IntegerArray ([1150](#)) type

PInterface = PUnknown

Pointer to IInterface ([1150](#)) interface

pinterfaceentry = ^tinterfaceentry

Pointer to tinterfaceentry ([1167](#)) record.

pinterfacetable = ^tinterfacetable

Pointer to tinterfacetable ([1167](#)) record.

PIntPtr = PPtrInt

Pointer to IntPtr (1150) type

PJump_buf = ^jump_buf

Pointer to jmp_buf (1150) record

PLongBool = ^LongBool

Pointer to a LongBool type.

PLongint = ^LongInt

Pointer to Longint (1150) type

PLongWord = ^LongWord

Pointer to LongWord type

PMemoryManager = ^TMemoryManager

Pointer to TMemoryManager (1168) record

PMsgStrTable = ^TMsgStrTable

Pointer to array of TMsgStrTable (1168) records.

PNativeInt = ^NativeInt

Pointer to NativeInt (1151) type

PNativeUInt = ^NativeUInt

Pointer to NativeInt (1151) type

PointerArray = Array[0..512*1024*1024-2] of Pointer

Generic pointer array.

POleVariant = ^OleVariant

Pointer to OleVariant type.

PPAnsiChar = PPChar

Alias for PPChar (1155) type.

PPByte = ^PByte

PPByte is a pointer to a PByte (1151) type.

PPChar = ^PChar

Pointer to an array of pointers to null-terminated strings.

```
PPCharArray = ^TPCharArray
```

Pointer to TPCharArray (1168) type.

```
PPCodePointer = ^PCodePointer
```

PPCodePointer is a typed pointer to PCodePointer (1152).

```
PPDispatch = ^PDispatch
```

Pointer to PDispatch (1152) pointer type

```
PPDouble = ^PDouble
```

PPDouble is a pointer to a PDouble (1152) type.

```
PPLongint = ^PLongint
```

PPLongint is a pointer to a PLongint (1154) type.

```
PPPointer = ^Pointer
```

Pointer to a pointer type.

```
PPPointerArray = ^PointerArray
```

Pointer to PointerArray (1154) type

```
PPPAansiChar = PPPChar
```

PPPAansiChar is a typed pointer to PPAnsichar (1154).

```
PPPChar = ^PPChar
```

PPPChar is a pointer to a PPChar (1155)

```
PPPointer = ^PPPointer
```

Pointer to a PPointer (1155) type.

```
PPPWideChar = ^PPWideChar
```

PPPWideChar is a pointer to a PPWideChar (1156) type.

```
PPPtrInt = ^PtrInt
```

Pointer to PtrInt (1157) type.

```
PPPtrUInt = ^PtrUInt
```


Pointer to unsigned integer of pointer size

`PUnknown = ^PUnknown`

Pointer to untyped pointer

`PPWideChar = ^PPWideChar`

Pointer to link id="PPWideChar"> type.

`PQWord = ^QWord`

Pointer to QWord type

`PRTL_CRITICAL_SECTION = ^RTL_CRITICAL_SECTION`

Pointer to #rtl.system.RTL_CRITICAL_SECTION (1169) type.

`RTL_EVENT = pointer`

Pointer to RTL_EVENT, which is an opaque type.

`PShortInt = ^ShortInt`

Pointer to shortint (1159) type

`PShortString = ^ShortString`

Pointer to a shortstring type.

`PSingle = ^Single`

Pointer to single-sized float value.

`PSizeInt = ^SizeInt`

Pointer to a SizeInt (1159) type

`PSmallInt = ^SmallInt`

Pointer to smallint (1159) type

`pstringmessageTable = ^TStringMessageTable`

Pointer to TStringMessageTable (1172) record.

`PText = ^Text`

Pointer to text file.

`PtInt = LongInt`

`Ptprint` is a signed integer type which has always the same size as a pointer. `Ptprint` 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(ptprint(p)+1024); {Overflow error.}
writeln(q>p);             {Incorrect answer.}
```

`Ptprint` 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 `ptprint` causes trouble in case the distance is larger than `high(ptprint)` and must be used with great care.

The introduction of the `ptprint` type was a mistake. Please use `ptruint` ([1157](#)) 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.

```
PUCS2Char = PWideChar
```

Pointer to `UCS2Char` ([1178](#)) character.

```
PUCS4Char = ^UCS4Char
```

Pointer to `UCS4Char` ([1178](#))

```
PUCS4CharArray = ^TUCS4CharArray
```

Pointer to array of `UCS4Char` ([1178](#)) characters.

```
pUInt16 = PWord
```

Pointer to `UInt16` ([1178](#)) type

```
pUInt32 = PDWord
```

Pointer to `UInt32` ([1178](#)) type

```
pUInt8 = PByte
```

Pointer to `UInt8` ([1179](#)) type

```
PUIntPtr = PPtrUInt
```

Pointer to `UIntPtr` ([1179](#)) 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

PUTF8String = ^UTF8String

Pointer to UTF8String (1179)

pvararray = ^tvararray

Pointer to TVarArray (1175) type.

pvararraybound = ^tvararraybound

Pointer to tvararraybound (1175) type.

pvararrayboundarray = ^tvararrayboundarray

Pointer to tvararrayboundarray (1175) type.

pvararraycoorarray = ^tvararraycoorarray

Pointer to tvararraycoorarray (1175) type.

pvardata = ^tvardata

Pointer to TVarData (1175) record.

PVariant = ^Variant

Pointer to Variant type.

pvariantmanager = ^tvariantmanager

Pointer to TVariantManager (1176) record.

PVarRec = ^TVarRec

Pointer to TVarRec (1177) type.

PVmt = ^TVmt

Pointer to TVMT (1178) record

PWideChar = ^WideChar

Pointer to WChar (1179).

`PWideString = ^WideString`

Pointer to widestring type

`PWord = ^Word`

Pointer to word (1179) 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 assigned it. Assigning a single-byte character string to this kind of string will not change the codepage of the string.

`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 = LongInt`

Signed integer type which fits for sizes

`SizeUInt = DWord`

Unsigned Integer type which fits for sizes

`Smallint = -32768..32767`

A signed 16-bits integer

`TAbstractErrorProc = procedure`

Abstract error handler procedural type.

`TAllocateThreadVarsHandler = procedure`

Threadvar allocation callback type for TThreadManager ([1173](#)).

```
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 ([1173](#)).

```
TBasicEventHandler = procedure(state: PEventState)
```

Generic callback type for handling eventstate in TThreadManager ([1173](#)).

```
TBasicEventWaitForHandler = function(timeout: Cardinal;
                                     state: PEventState) : LongInt
```

Wait for basic event callback type for TThreadManager ([1173](#)).

```
TBeginThreadHandler = function(sa: Pointer; stacksize: PtrUInt;
                              ThreadFunction: TThreadFunc; p: pointer;
                              creationFlags: DWord;
                              var ThreadId: TThreadID) : TThreadID
```

Callback for thread start in TThreadManager ([1173](#)).

```
TBoundArray = Array of SizeInt
```

Dynamic array of integer.

```
tcalldesc = packed record
  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` ([1349](#)).

```
TCriticalSectionHandler = procedure(var cs)
```

Generic callback type for critical section handling in `TThreadManager` ([1173](#)).

```
TCriticalSectionHandlerTryEnter = function(var cs) : LongInt
```

`TCriticalSectionHandlerTryEnter` is the function prototype for the `TryEnterCriticalSection` ([1330](#)) function, in the `TThreadManager` ([1173](#)) 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` ([1161](#)) instead.

```
TDateTime = Double
```

Encoded Date-Time type.

```
tdispdesc = packed record
  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.

```
TDoubleRec = packed record
  function GetExp : QWord;
  procedure SetExp(e: QWord);
  function GetSign : Boolean;
  procedure SetSign(s: Boolean);
  function GetFrac : QWord;
  procedure SetFrac(e: QWord);
  function Mantissa : QWord;
  function Fraction : ValReal;
  function Exponent : LongInt;
```

```

    Sign : Boolean;
    Exp : QWord;
    Frac : QWord;
    function SpecialType : TFloatSpecial;
end

```

TDoubleRec models the memory layout of a double value when using software floating point math.

```
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
    kind : Byte;
    namelen : Byte;
    elesize : SizeInt;
    eletype : pdynarraytypeinfo;
    vartype : LongInt;
end

```

`tdynarraytypeinfo` describes the structure of a multi-dimensional dynamical array. It is used in the `DynArraySetLength` ([1212](#)) call.

```
TEndThreadHandler = procedure(ExitCode: DWord)
```

Callback for thread end in `TThreadManager` ([1173](#)).

```

TEntryInformation = record
    InitFinalTable : Pointer;
    ThreadvarTablesTable : Pointer;
    asm_exit : procedure;
    PascalMain : procedure;
    valgrind_used : Boolean;
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.

```
TError = LongInt
```

Error type, used in variants.

```

TErrorProc = procedure(ErrNo: LongInt;Address: CodePointer;
                      Frame: Pointer)

```

Standard error handler procedural type.

```
TExceptObject = record
  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 ([1164](#)).

```
TExtended80Rec = packed record
  function GetExp : QWord;
  procedure SetExp(e: QWord);
  function GetSign : Boolean;
  procedure SetSign(s: Boolean);
  function Mantissa : QWord;
  function Fraction : Extended;
  function Exponent : LongInt;
  Sign : Boolean;
  Exp : QWord;
  function SpecialType : TFloatSpecial;
end
```

TExtended80Rec models the memory layout of an extended value when using software floating point math.

```
TextFile = Text
```

Alias for Text file type.

```
TextRec = record
  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` (1164) or `FileRec` (1149) 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 36.10: Enumeration values for type `TFloatSpecial`

Value	Explanation
<code>fsDenormal</code>	Denormal value
<code>fsInf</code>	Infinity
<code>fsInvalidOp</code>	Invalid operation
<code>fsNaN</code>	Not a number
<code>fsNDenormal</code>	Negative enormal value
<code>fsNegative</code>	Negative value
<code>fsNInf</code>	Negative infinity
<code>fsNZero</code>	Negative zero
<code>fsPositive</code>	Positive value
<code>fsZero</code>	Zero

`TFloatSpecial` enumerates a series of floating point value properties.

```

TFPCHeapStatus = record
  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` (1166) record defined by Delphi, which contains information not meaningful for the FPC heap manager. The heap status can be retrieved by the `GetFPCHeapStatus` (1230) 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 36.11: Enumeration values for type `TFPUException`

Value	Explanation
<code>exDenormalized</code>	
<code>exInvalidOp</code>	Invalid operation error
<code>exOverflow</code>	Float overflow error
<code>exPrecision</code>	Precision error
<code>exUnderflow</code>	Float underflow error
<code>exZeroDivide</code>	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 36.12: Enumeration values for type `TFPUPrecisionMode`

Value	Explanation
<code>pmDouble</code>	Double-type precision
<code>pmExtended</code>	Extended-type precision
<code>pmReserved</code>	?
<code>pmSingle</code>	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 36.13: 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 (1173).

```
TGuid = packed record
end
```

Standard GUID representation type.

```
THandle = LongInt
```

This type should be considered opaque. It is used to describe file and other handles.

```
THeapStatus = record
  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 (1231) call.

```
TInitThreadVarHandler = procedure (var offset: DWord; size: DWord)
```

Threadvar initialization callback type for TThreadManager (1173).

```
TInterfacedClass = Class of TInterfacedObject
```

TInterfacedClass is a descendent of

```
tinterfaceentry = record
  IID : PGuid;
  VTable : Pointer;
  IOffset : SizeUInt;
  IIDStr : PShortString;
end
```

tinterfaceentry is used to store the list of Interfaces of a class. This list is stored as an array of tinterfaceentry records.

```
tinterfaceentrytype = (etStandard, etVirtualMethodResult,
  etStaticMethodResult, etFieldValue,
  etVirtualMethodClass, etStaticMethodClass,
  etFieldValueClass)
```

Table 36.14: 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
  EntryCount : SizeUInt;
  Entries : Array[0..0] of tinterfaceentry;
end
```

Record to store list of interfaces of a class.

```
TLineEndStr = string
```

TLineEndStr is an alias for the actual line ending string type, used in TextRec (1164). It should be treated as opaque.

```
TMemoryManager = record
  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;
GetFPCHHeapStatus : function : TFPCHHeapStatus;
end

```

TMemoryManager describes the memory manager. For more information about the memory manager, see the programmer's reference.

```

TMethod = record
  Code : CodePointer;
  Data : Pointer;
end

```

TMethod describes a general method pointer, and is used in Run-Time Type Information handling.

```

TMsgStrTable = record
  name : PShortString;
  method : CodePointer;
end

```

Record used in string message handler table.

```

TPCharArray = packed Array[0..(MaxLongintdivSizeOf(PChar))-1] of PChar

```

Array of PChar

```

TProcedure = procedure

```

Simple procedural type.

```

TReleaseThreadVarsHandler = procedure

```

Threadvar release callback type for TThreadManager ([1173](#)).

```

TRelocateThreadVarHandler = function(offset: DWord) : pointer

```

Threadvar relocation callback type for TThreadManager ([1173](#)).

```

TResourceHandle = PtrUInt

```

This is an opaque type.

```

TResourceManager = record
  HINSTANCEFunc : function : TFPResourceHMODULE;
  EnumResourceTypesFunc : function(ModuleHandle: TFPResourceHMODULE;EnumFunc: EnumRe

```

```

        lParam: PtrInt) : LongBool;
EnumResourceNamesFunc : function(ModuleHandle: TFPResourceHMODULE; ResourceType: PC
        EnumFunc: EnumResNameProc; lParam: PtrInt) : LongBool;
EnumResourceLanguagesFunc : function(ModuleHandle: TFPResourceHMODULE; ResourceType
        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: TFPResou
        : TFPResourceHGLOBAL;
SizeofResourceFunc : function(ModuleHandle: TFPResourceHMODULE; ResHandle: TFPResou
        : 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 ([1173](#)).

```
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 ([1173](#)).

```
TRTLEventHandlerTimeout = procedure(AEvent: PRTLEvent; timeout: LongInt)
```

TRTLEvent timeout handling type for TThreadManager ([1173](#)).

```
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)
```

Table 36.15: 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
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
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 (1217) 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.

```
TSingleRec = packed record
  function GetExp : QWord;
  procedure SetExp(e: QWord);
  function GetSign : Boolean;
  procedure SetSign(s: Boolean);
  function GetFrac : QWord;
  procedure SetFrac(e: QWord);
  function Mantissa : QWord;
  function Fraction : ValReal;
  function Exponent : LongInt;
  Sign : Boolean;
  Exp : QWord;
  Frac : QWord;
  function SpecialType : TFloatSpecial;
end
```

TsingleRec models the memory layout of a double value when using software floating point math.

```
TStandardCodePageEnum = (scpAnsi, scpConsoleInput, scpConsoleOutput,
  scpFileSystemSingleByte)
```

Table 36.16: 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 ([1174](#)).


```
TStringMessageTable = record
  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 (1168) 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 36.17: 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 (1173).

```
TThreadHandler = function(threadHandle: TThreadID) : DWord
```

Generic thread handler callback for TThreadManager (1173).

```
TThreadID = PtrUInt
```

This is an opaque type, it can differ from operating system to operating system.

```
TThreadManager = record
  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;
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;
SemaphoreInit : TSemaphoreInitHandler;
SemaphoreDestroy : TSemaphoreDestroyHandler;
SemaphorePost : TSemaphorePostHandler;
SemaphoreWait : TSemaphoreWaitHandler;
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` (1313) procedure, and the current thread manager can be retrieved with the `GetThreadManager` (1233) 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` (1173).

```

TThreadSwitchHandler = procedure

```

Callback type for thread switch in TThreadManager (1173).

TTime = TDateTime

TTime is defined for Delphi compatibility. This type is deprecated, use TDateTime (1161) instead.

TUCS4CharArray = Array[0..\$effffff] of UCS4Char

Array of UCS4Char (1178) characters.

```
TUnicodeStringManager = record
  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) : PtrInt;
  CompareTextWideStringProc : function(const s1: WideString;const s2: WideString) : PtrInt;
  CharLengthPCharProc : function(const Str: PChar) : PtrInt;
  CodePointLengthProc : function(const Str: PChar;MaxLookAhead: 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) : PtrInt;
  CompareTextUnicodeStringProc : function(const s1: UnicodeString;const s2: UnicodeString) : PtrInt;
  GetStandardCodePageProc : function(const stdcp: TStandardCodePageEnum) : TSystemCodePage;
end
```

TUnicodeStringManager is currently the same as the TUnicodeStringManager (1174) manager record. It performs the same functions: converting unicode strings to ansistrings and vice-versa, performing uppercase to lowercase transformations and comparing strings.

```
tvararray = record
  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 (1175) records equal to the number of dimensions in the array (dimcoun).

```

tvararraybound = record
    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 (1175) records.

```
tvararraycoorarray = Array[0..0] of LongInt
```

Array of variant array coordinates

```

tvardata = packed record
    vtype : tvartype;
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
    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: widestring; 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: poin
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: LongInt;
    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 (1314) call.

```

tvarop = (opadd, opsubtract, opmultiply, opdivide, opintdivide, opmodulus,
    opshiftleft, opshiftright, opand, opor, opxor, opcompare, opnegate,
    opnot, opcmpeq, opcmpne, opcmplt, opcmple, opcmpgt, opcmpge, oppower)

```

Table 36.18: 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
opshiftright	Variant operation: Shift left
opshiftright	Variant operation: Shift right
opsubtract	Variant operation: Substraction
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
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.

```
TVMt = record
  vInstanceSize : SizeInt;
  vInstanceSize2 : SizeInt;
  vParent : PVmt;
  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;
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.

```

TWaitForThreadTerminateHandler = function(threadHandle: TThreadID;
                                          TimeoutMs: LongInt) : DWord

```

Callback type for thread termination in TThreadManager ([1173](#)).

```

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 ([1178](#)) 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` (1157) type for compatibility with later Delphi versions.

```
UnicodeChar = WideChar
```

`UnicodeChar` is a single character from a `UnicodeString`. It equals `WideChar` in all respects.

```
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` (1336) function.

```
ValUInt = Cardinal
```

Integer with the same size as the return code of the `Val` (1336) 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` (1118) and `WideString` (1118)

```
Word = 0..65535
```

An unsigned 16-bits integer

36.10.3 Variables

```
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` (1181). 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` (1181).

The value of this variable may be changed using the `SetMultiByteFileSystemCodePage` (1310) 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` (1159) file/path RTL routines.

Examples include the file/path names returned by the `RawByteString` versions of `SysUtils.FindFirst` (1118) and `GetDir` (1230).

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` (1181)).

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` (1180). 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` (1310) 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` (1309). That means that it is not a good idea to use its value to determine the real OS "default system code page".

`DefaultUnicodeCodePage : TSystemCodePage`

`DefaultUnicodeCodePage` is the unicode code page for a new unicode string. On most platforms, this is `CP_UTF16` (1125).

`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.

`ErrOutput : Text`

`ErrOutput` is provided for Delphi compatibility.

`ExitCode : LongInt; public name '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

Result of last I/O operation. Read-Only.

Input : Text

Standard input text file.

IsConsole : Boolean; public name 'operatingsystem_isconsole' = False

True for console applications, False for GUI applications.

IsLibrary : Boolean; public name '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. Its use is not recommended, and it is not even available on other platforms.

meml : Array[0..(\$7fffffffdivsizeof(longint))-1] of LongInt

meml 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. Its 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. Its 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 ([1290](#)) 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` : `TFPUExceptionMask`

Current soft float exception flags

`softfloat_exception_mask` : `TFPUExceptionMask`

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 ([1182](#)).

`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.

36.11 Procedures and functions

36.11.1 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` ([1300](#))

Listing: `./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.
```

36.11.2 AbstractError

Synopsis: Generate an abstract error.

Declaration: `procedure AbstractError`

Visibility: default

Description: `AbstractError` generates an abstract error (run-time error 211). If the `AbstractErrorProc` ([1123](#)) constant is set, it will be called instead.

Errors: This routine causes a run-time error 211.

See also: `AbstractErrorProc` ([1123](#))

36.11.3 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` (1294) 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` (1294)

36.11.4 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` (1126)

36.11.5 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` (1315)

Listing: `./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**(1); *{ P points to 1 }*
 p:=**Addr**(Zero); *{ P points to 'Zero' }*
end.

36.11.6 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.

36.11.7 AllocMem

Synopsis: Allocate and clear memory.

Declaration: `function AllocMem (Size: PtrUInt) : pointer`

Visibility: default

Description: `AllocMem` calls `getmem GetMem` ([1231](#)), and clears the allocated memory, i.e. the allocated memory is filled with `Size` zero bytes.

See also: `GetMem` ([1231](#))

36.11.8 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` ([1335](#))

36.11.9 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 ([1296](#)), Close ([1198](#)), Reset ([1295](#))

Listing: ./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 }*

WriteIn (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. }*

WriteIn (f, 'This is the second line of text.txt');

close (f);

end.

36.11.10 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 ([1315](#)), Cos ([1206](#))

Listing: ./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.

36.11.11 ArrayStringToPPchar

Synopsis: Concert 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` ([1321](#))

36.11.12 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 global 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` ([1235](#)), `Runerror` ([1302](#))

36.11.13 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.

Errors: None.

See also: [Reset \(1295\)](#), [Rewrite \(1296\)](#), [Append \(1186\)](#)

Listing: ./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.

36.11.14 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 \(1256\)](#)

Listing: ./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.

36.11.15 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` ([1301](#)) instead.

See also: `RTLEventCreate` ([1301](#))

36.11.16 basiceventdestroy

Synopsis: Obsolete. Don't use

Declaration: `procedure basiceventdestroy(state: PEventState)`

Visibility: default

Description: `basiceventdestroy` is obsolete. Use `RTLEventDestroy` ([1301](#)) instead.

See also: `RTLEventDestroy` ([1301](#))

36.11.17 basiceventResetEvent

Synopsis: Obsolete. Don't use

Declaration: `procedure basiceventResetEvent(state: PEventState)`

Visibility: default

Description: `basiceventResetEvent` is obsolete. Use `RTLEventResetEvent` ([1301](#)) instead.

See also: `RTLEventResetEvent` ([1301](#))

36.11.18 basiceventSetEvent

Synopsis: Obsolete. Don't use

Declaration: `procedure basiceventSetEvent(state: PEventState)`

Visibility: default

Description: `basiceventSetEvent` is obsolete. Use `RTLEventSetEvent` ([1302](#)) instead.

See also: `RTLEventSetEvent` ([1302](#))

36.11.19 basiceventWaitFor

Synopsis: Obsolete. Don't use

Declaration: `function basiceventWaitFor(Timeout: Cardinal;state: PEventState)
 : LongInt`

Visibility: default

Description: `basiceventwaitfor` is obsolete. Use `RTLEventWaitFor` ([1302](#)) instead.

See also: `RTLEventWaitFor` ([1302](#))

36.11.20 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 `\ostwo`) 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` ([1213](#))

36.11.21 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` ([1251](#)), `NtoBE` ([1257](#)), `NtoLE` ([1257](#))

36.11.22 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` ([1319](#)), `Val` ([1336](#)), `HexStr` ([1236](#)), `OctStr` ([1257](#))

Listing: `./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.
```

36.11.23 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` ([1296](#)) or `Reset` ([1295](#)) 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` ([1193](#)), `Close` ([1198](#)), `Reset` ([1295](#)), `Assign` ([1188](#))

Listing: `./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.

```

36.11.24 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` (1296) or `Reset` (1295) 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` (1192), `Close` (1198), `Rewrite` (1296), `Assign` (1188)

36.11.25 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`, `var{repeat}` and `While` statements.

Note that while this is a procedure, `Break` is a reserved word and hence cannot be redefined.

Errors: None.

See also: `Continue` ([1204](#)), `Exit` ([1219](#))

Listing: `./refex/ex87.pp`

Program `Example87`;

{ Program to demonstrate the Break function. }

```

Var I : longint;

begin
  I:=0;
  While I<10 Do
    begin
      Inc(I);
      If I>5 Then
        Break;
      Writeln (i);
    end;
  I:=0;
  Repeat
    Inc(I);
    If I>5 Then
      Break;
    Writeln (i);
  Until I>=10;
  For I:=1 to 10 do
    begin
      If I>5 Then
        Break;
      Writeln (i);
    end;
end.
```

36.11.26 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.

See also: `BsrByte` ([1195](#)), `BsfWord` ([1195](#)), `BsfDWord` ([1195](#)), `BsfQWord` ([1195](#))

36.11.27 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.

See also: `BsfByte` ([1194](#)), `BsfWord` ([1195](#)), `BsrDWord` ([1196](#)), `BsfQWord` ([1195](#))

36.11.28 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.

See also: `BsfByte` ([1194](#)), `BsfWord` ([1195](#)), `BsfDWord` ([1195](#)), `BsrQWord` ([1196](#))

36.11.29 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.

See also: `BsfByte` ([1194](#)), `BsrWord` ([1196](#)), `BsfDWord` ([1195](#)), `BsfQWord` ([1195](#))

36.11.30 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: `BsrByte` 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.

See also: `BsfByte` ([1194](#)), `BsrWord` ([1196](#)), `BsrDWord` ([1196](#)), `BsrQWord` ([1196](#))

36.11.31 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.

See also: `BsrByte` (1195), `BsrWord` (1196), `BsfDWord` (1195), `BsrQWord` (1196)

36.11.32 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.

See also: `BsfByte` (1194), `BsfWord` (1195), `BsfDWord` (1195), `BsrQWord` (1196)

36.11.33 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.

See also: `BsrByte` (1195), `BsfWord` (1195), `BsrDWord` (1196), `BsrQWord` (1196)

36.11.34 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` (1235), `get_caller_stackinfo` (1234), `get_caller_addr` (1234), `get_caller_frame` (1234)

36.11.35 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` ([1255](#)), `Rmdir` ([1297](#))

Listing: `./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.
```

36.11.36 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` ([1284](#)), `Str` ([1319](#))

Listing: `./refex/ex8.pp`

Program `Example8;`

{ Program to demonstrate the Chr function. }

```
begin
  Write (chr(10),chr(13)); { The same effect as Writeln; }
end.
```

36.11.37 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` (1295) or `Rewrite` (1296) is sufficient.

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: `Assign` (1188), `Reset` (1295), `Rewrite` (1296), `Flush` (1226)

Listing: `./refex/ex9.pp`

Program `Example9;`

{ Program to demonstrate the Close function. }

Var `F : text;`

begin

`Assign (f, 'Test.txt');`

ReWrite (`F`);

WriteLn (`F`, 'Some text written to Test.txt');

`close (f); { Flushes contents of buffer to disk,`
closes the file. Omitting this may
cause data NOT to be written to disk. }

end.

36.11.38 CloseThread

Synopsis: Close a thread and free up resources used by the thread

Declaration: `function CloseThread(threadHandle: TThreadID) : DWord`

Visibility: default

Description: `CloseThread` must be called on any thread started with `BeginThread` (1191). It must be called after the thread has ended (either by exiting the thread function or after calling `EndThread` (1213)).

Errors: If no threadmanager is installed, an exception may be raised or runtime error 232 may occur if no exceptions are used.

See also: `BeginThread` (1191), `EndThread` (1213)

36.11.39 CompareByte

Synopsis: Compare 2 memory buffers byte per byte

Declaration: `function CompareByte(const buf1;const buf2;len: SizeInt) : SizeInt`

Visibility: default

Description: CompareByte compares two memory regions buf1,buf2 on a byte-per-byte basis for a total of len bytes.

The function returns one of the following values:

less than 0 if buf1 and buf2 contain different bytes in the first len bytes, and the first such byte is smaller in buf1 than the byte at the same position in buf2.

0 if the first len bytes in buf1 and buf2 are equal.

greater than 0 if buf1 and buf2 contain different bytes in the first len bytes, and the first such byte is larger in buf1 than the byte at the same position in buf2.

Errors: None.

See also: CompareChar ([1199](#)), CompareChar0 ([1201](#)), CompareWord ([1202](#)), CompareDWord ([1201](#))

Listing: ./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.

36.11.40 CompareChar

Synopsis: ompare 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:

-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`.

0If the first `len` characters in `buf1` and `buf2` are equal.

1If `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` ([1198](#)), `CompareChar0` ([1201](#)), `CompareWord` ([1202](#)), `CompareDWord` ([1201](#))

Listing: `./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 - 1);
    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.

```

36.11.41 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 ([1198](#)), CompareChar ([1199](#)), CompareDWord ([1201](#)), CompareWord ([1202](#))

36.11.42 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:

- 1 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.
- 1 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 ([1199](#)), CompareByte ([1198](#)), CompareWord ([1202](#))

Listing: ./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.

36.11.43 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:

- 1 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.

0if the first `len` Words in `buf1` and `buf2` are equal.

1if `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` ([1199](#)), `CompareByte` ([1198](#)), `CompareDWord` ([1201](#))

Listing: `./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.

36.11.44 Concat

Synopsis: Append one string 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 resulting string is truncated at a length of 255 bytes. The same operation can be performed with the `+` operation.

Errors: None.

See also: Copy ([1205](#)), Delete ([1208](#)), Insert ([1244](#)), Pos ([1287](#)), Length ([1250](#))

Listing: ./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.
```

36.11.45 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 while this is a procedure, `Continue` is a reserved word and hence cannot be redefined.

Errors: None.

See also: Break ([1193](#)), Exit ([1219](#))

Listing: ./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.

```

36.11.46 Copy

Synopsis: Copy part of a string.

Declaration: `function Copy(S: AStringType; Index: Integer; Count: Integer) : string`
`function Copy(A: DynArrayType; Index: Integer; Count: Integer) : DynArray`

Visibility: default

Description: Copy returns a string which is a copy of 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 dynamical arrays, Copy returns a new dynamical array of the same type as the original one, and copies Count elements from the old array, starting at position Index.

Errors: None.

See also: Delete ([1208](#)), Insert ([1244](#)), Pos ([1287](#))

Listing: ./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.

36.11.47 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 \(1244\)](#), [FinalizeArray \(1224\)](#), [DynArraySize \(1213\)](#), [DynArrayClear \(1212\)](#), [DynArrayDim \(1212\)](#), [DynArrayBounds \(1211\)](#)

36.11.48 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 2π , then the result is undefined.

Errors: None.

See also: [Arctan \(1187\)](#), [Sin \(1315\)](#)

Listing: `./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.

36.11.49 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 \(1210\)](#), [Seg \(1306\)](#), [Ofs \(1258\)](#), [Ptr \(1289\)](#)

Listing: `./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.

36.11.50 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.

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` ([1238](#))

Listing: `./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.

36.11.51 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.

36.11.52 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` ([1208](#))

36.11.53 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` ([1183](#))

36.11.54 Delete

Synopsis: Delete part of a string.

Declaration: `procedure Delete(var s: shortstring; index: SizeInt; count: SizeInt)
procedure Delete(var S: RawByteString; Index: SizeInt; Size: SizeInt)
procedure Delete(var S: UnicodeString; Index: SizeInt; Size: SizeInt)
procedure Delete(var S: WideString; Index: SizeInt; Size: SizeInt)`

Visibility: default

Description: `Delete` removes `Count` characters from string `S`, starting at position `Index`. All characters after the deleted characters are shifted `Count` positions to the left, and the length of the string is adjusted.

Errors: `Shortstring` variant's third parameter is called `Count`, in other overloaded variants it is called `Index`

See also: `Copy` ([1205](#)), `Pos` ([1287](#)), `Insert` ([1244](#))

Listing: `./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.

```

36.11.55 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` ([1256](#)). 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` ([1256](#)), `Getmem` ([1231](#)), `Freemem` ([1228](#))

Listing: `./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.

```

36.11.56 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` (1214) or `LeaveCriticalsection` (1250), unless it is again initialized with `InitCriticalSection` (1243)

See also: `InitCriticalSection` (1243), `EnterCriticalSection` (1214), `LeaveCriticalSection` (1250)

36.11.57 DoneThread

Synopsis: End the current thread

Declaration: `procedure DoneThread`

Visibility: default

Description: `DoneThread` should be used to end the current thread. It performs the necessary housekeeping before actually ending the thread. Using the operating system calls to end the thread may result in data corruption or memory leaks.

See also: `BeginThread` (1191)

36.11.58 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` (1206), `Seg` (1306), `Ofs` (1258), `Ptr` (1289)

Listing: ./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.

36.11.59 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` ([1211](#))

36.11.60 Dump_Stack

Synopsis: Dump stack to the given text file.

Declaration: `procedure Dump_Stack(var f: text; fp: pointer; addr: CodePointer)`

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` ([1234](#)), `get_caller_frame` ([1234](#)), `get_frame` ([1235](#))

36.11.61 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` ([1244](#)), `FinalizeArray` ([1224](#)), `CopyArray` ([1205](#)), `DynArraySize` ([1213](#)), `DynArrayClear` ([1212](#)), `DynArrayDim` ([1212](#))

36.11.62 DynArrayClear

Synopsis: Clear dynamical 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 \(1244\)](#), [FinalizeArray \(1224\)](#), [CopyArray \(1205\)](#), [DynArraySize \(1213\)](#), [DynArrayDim \(1212\)](#), [DynArrayBounds \(1211\)](#)

36.11.63 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 \(1244\)](#), [FinalizeArray \(1224\)](#), [CopyArray \(1205\)](#), [DynArraySize \(1213\)](#), [DynArrayClear \(1212\)](#), [DynArrayBounds \(1211\)](#)

36.11.64 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 typinfo. The length of indices must equal the number of dimensions of the array (as returned by `DynArrayDim (1212)`).

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 \(1244\)](#), [FinalizeArray \(1224\)](#), [CopyArray \(1205\)](#), [DynArraySize \(1213\)](#), [DynArrayClear \(1212\)](#), [DynArrayBounds \(1211\)](#), [DynArrayDim \(1212\)](#), [IsDynArrayRectangular \(1249\)](#)

36.11.65 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 dynamical array `a` to the first `dimCnt` lengths specified in the array `lengthVec`. The dynamical array type is described in `typeInfo` which points to a record of type `TDynArrayTypeInfo` (1162)

It should never be necessary to call this function directly, the standard `SetLength` (1309) function should be used instead.

Errors: If an invalid pointer is specified, an error may occur.

See also: `SetLength` (1309), `tdynarraytypeinfo` (1162)

36.11.66 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` (1250) for dynamic arrays.

See also: `InitializeArray` (1244), `FinalizeArray` (1224), `CopyArray` (1205), `DynArrayClear` (1212), `DynArrayDim` (1212), `DynArrayBounds` (1211)

36.11.67 EmptyMethod

Synopsis: Empty method alias

Declaration: `procedure EmptyMethod`

Visibility: default

Description: `EmptyMethod` is meant for the compiler only. It should not be used directly.

36.11.68 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` (1338)). If it is omitted, zero is used.

This function does not return.

See also: `WaitForThreadTerminate` (1338), `BeginThread` (1191)

36.11.69 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` (1250)), 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` (1243) prior to a call to `EnterCriticalSection`.

A call to `EnterCriticalSection` must always be matched by a call to `LeaveCriticalSection` (1250). 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` (1243), `DoneCriticalSection` (1210), `LeaveCriticalSection` (1250)

36.11.70 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` (1215), `EnumResourceNames` (1215), `EnumResourceLanguages` (1214)

36.11.71 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` ([1215](#)), `EnumResourceLanguages` ([1214](#))

36.11.72 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` ([1215](#)), `EnumResourceLanguages` ([1214](#))

36.11.73 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 `{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: Eoln ([1216](#)), Assign ([1188](#)), Reset ([1295](#)), Rewrite ([1296](#))

Listing: ./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.
```

36.11.74 EOLn

Synopsis: Check for end of line

Declaration: function EOLn(var t: Text) : Boolean
 function EOLn : Boolean

Visibility: default

Description: Eof 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 Eof 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 ([1215](#)), Assign ([1188](#)), Reset ([1295](#)), Rewrite ([1296](#))

Listing: ./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.
```

36.11.75 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 `{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: `Assign` ([1188](#))

Listing: `./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.

36.11.76 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` ([1302](#)), `Halt` ([1235](#))

36.11.77 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 ([1239](#))

Listing: ./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.
```

36.11.78 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.

Errors: None.

See also: `Halt` ([1235](#))

Listing: `./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.

36.11.79 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 ([1251](#)), Power ([1288](#))

Listing: ./refex/ex22.pp

Program Example22;

{ Program to demonstrate the Exp function. }

begin

 Writeln (Exp(1):8:2); *{ Should print 2.72 }*

end.

36.11.80 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 ([1295](#)) isn't specified.

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: Filesize ([1221](#))

Listing: ./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.

```

36.11.81 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` ([1295](#)) 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: `Filepos` ([1220](#))

Listing: `./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.

```

36.11.82 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` ([1224](#)), and if it is a multiple of 4 bytes it is better to use `FillDWord` ([1223](#)), these routines are optimized for their respective sizes.

Errors: No checking on the size of `X` is done.

See also: `FillChar` ([1222](#)), `FillDWord` ([1223](#)), `FillWord` ([1224](#)), `Move` ([1255](#))

Listing: `./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.

```

36.11.83 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 ([1224](#)), Move ([1255](#)), FillByte ([1222](#)), FillDWord ([1223](#))

Listing: ./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.

```

36.11.84 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 ([1222](#)), Fillchar ([1222](#)), Fillword ([1224](#)), Move ([1255](#))

Listing: ./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.

```

36.11.85 FillQWord

Synopsis: Fill memory range with QWord (64-bit) values

Declaration: `procedure FillQWord(var x; count: SizeInt; value: QWord)`

Visibility: default

Description: `FillQWord` fills the memory location of `x` with `Count` times `value`. The size of the filled memory location is therefor `8*count` bytes.

Errors: No checks are made to see if `X` actually has a minimum size of `(Count*8)` bytes. Therefor, other variables can be overwritten or the memory may be out of the accessible memory for the program. In the latter case a run-error or exception may be triggered.

See also: `FillChar` ([1222](#)), `FillWord` ([1224](#))

36.11.86 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` ([1222](#)), `Move` ([1255](#))

Listing: `./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.
```

36.11.87 FinalizeArray

Synopsis: Finalize managed-type elements in array

Declaration: `procedure FinalizeArray(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: `InitializeArray` ([1244](#)), `CopyArray` ([1205](#)), `DynArraySize` ([1213](#)), `DynArrayClear` ([1212](#)), `DynArrayDim` ([1212](#)), `DynArrayBounds` ([1211](#))

36.11.88 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;
                    ResourceName: AnsiString; 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` (1252).

Errors: None. In case the resource was not found, 0 is returned.

See also: `FreeResource` (1229), `LoadResource` (1252), `SizeofResource` (1316), `LockResource` (1253), `UnlockResource` (1333), `FreeResource` (1229)

36.11.89 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;
                      ResourceType: AnsiString;
                      ResourceName: AnsiString; 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.

36.11.90 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` (1183).

See also: `softfloat_exception_flags` (1183), `softfloat_exception_mask` (1183)

36.11.91 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 `\textit{not}` closed as a result of this call.

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: `Close` (1198)

Listing: `./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.

36.11.92 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` ([1213](#))

36.11.93 FMADouble

Synopsis: Internal function, do not use

Declaration: `function FMADouble(d1: Double; d2: Double; d3: Double) : Double`

Visibility: default

36.11.94 FMAExtended

Synopsis: Internal function, do not use

Declaration: `function FMAExtended(e1: extended; e2: extended; e3: extended) : extended`

Visibility: default

36.11.95 FMASingle

Synopsis: Internal function, do not use

Declaration: `function FMASingle(s1: single; s2: single; s3: single) : single`

Visibility: default

36.11.96 fpc_dynarray_rangecheck

Synopsis: Auxiliary compiler routine

Declaration: `procedure fpc_dynarray_rangecheck(p: pointer; i: tdynarrayindex)`

Visibility: default

Description: `fpc_dynarray_rangecheck` is an auxiliary routine for checking whether an array index is in range for the array.

36.11.97 fpc_SarInt64

Synopsis: SAR (int64)

Declaration: `function fpc_SarInt64(const AValue: Int64; const Shift: Byte) : Int64`

Visibility: default

Description: `fpc_SarInt64` is a FPC helper function which should not be used directly.

36.11.98 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.

36.11.99 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 \(1300\)](#), [Int \(1245\)](#)

Listing: `./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.

36.11.100 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 \(1231\)](#), [New \(1256\)](#), [Dispose \(1209\)](#)

Listing: `./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 .

```

36.11.101 Freememory

Synopsis: Alias for FreeMem ([1228](#))

Declaration: `procedure Freememory(p: pointer;Size: PtrUInt)`
`function Freememory(p: pointer) : PtrUInt`

Visibility: default

Description: `FreeMemory` is an alias for `FreeMem` ([1228](#)).

See also: `FreeMem` ([1228](#))

36.11.102 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` ([1252](#)). It returns `True` if the operation was succesful, `False` otherwise.

Errors: On error, `False` is returned.

See also: `FindResource` ([1225](#)), `LoadResource` ([1252](#)), `SizeofResource` ([1316](#)), `LockResource` ([1253](#)), `UnlockResource` ([1333](#)), `FreeResource` ([1229](#))

36.11.103 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` ([1118](#))

36.11.104 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` ([1249](#)) or `ThreadSetPriority` ([1328](#))

Errors: None.

See also: `KillThread` ([1249](#)), `ThreadSetPriority` ([1328](#))

36.11.105 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` ([1197](#))

Listing: `./refex/ex29.pp`

Program `Example29;`

`{ Program to demonstrate the GetDir function. }`

`Var S : String;`

`begin`

`GetDir (0,S);`

`Writeln ('Current directory is : ',S);`

`end.`

36.11.106 GetFPCHeapStatus

Synopsis: Return FPC heap manager status information

Declaration: `function GetFPCHeapStatus : TFPCHeapStatus`

Visibility: default

Description: Return FPC heap manager status information

36.11.107 GetHeapStatus

Synopsis: Return the memory manager heap status.

Declaration: `function GetHeapStatus : THeapStatus`

Visibility: default

36.11.108 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` (1182): 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` (1222) or `FillWord` (1224).

For an example, see `Freemem` (1228).

Errors: None.

See also: `Freemem` (1228), `Dispose` (1209), `New` (1256), `returnnilifgrowheapfails` (1182)

36.11.109 GetMemory

Synopsis: Alias for `GetMem` (1231)

Declaration: `procedure Getmemory(out p: pointer; Size: PtrUInt)`
`function GetMemory(size: PtrUInt) : pointer`

Visibility: default

Description: `Getmemory` is an alias for `GetMem` (1231).

See also: `GetMem` (1231)

36.11.110 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` (1309), `IsMemoryManagerSet` (1249)

36.11.111 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` ([1232](#))

36.11.112 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` ([1169](#)), `SetResourceManager` ([1311](#))

36.11.113 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` ([1164](#)), `SetTextCodePage` ([1312](#))

36.11.114 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` ([1232](#))

36.11.115 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` ([1313](#)), `TThreadManager` ([1173](#))

36.11.116 GetUnicodeStringManager

Synopsis: Return a copy of the currently active unicodetring manager.

Declaration: `procedure GetUnicodeStringManager(var Manager: TUnicodeStringManager)`

Visibility: default

Description: `GetUnicodeStringManager` returns a copy of the currently active unicode string 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 `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` ([1313](#)), `TUnicodeStringManager` ([1174](#))

36.11.117 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` ([1314](#))

36.11.118 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 heap 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: `SetWideStringManager` ([1314](#)), `TWideStringManager` ([1178](#))

36.11.119 `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` ([1235](#)), `get_caller_frame` ([1234](#)), `Dump_Stack` ([1211](#))

36.11.120 `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` ([1234](#)), `get_frame` ([1235](#)), `Dump_Stack` ([1211](#))

36.11.121 `get_caller_stackinfo`

Synopsis: Return caller stack information

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` ([1196](#)), `Get_pc_addr` ([1235](#)), `get_caller_addr` ([1234](#)), `get_caller_frame` ([1234](#))

36.11.122 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` ([1285](#)) and `ParamStr` ([1285](#)) should be used.

See also: `ParamCount` ([1285](#)), `ParamStr` ([1285](#))

36.11.123 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` ([1234](#)), `get_caller_frame` ([1234](#))

36.11.124 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` ([1196](#)), `get_caller_stackinfo` ([1234](#)), `get_caller_addr` ([1234](#)), `get_caller_frame` ([1234](#))

36.11.125 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.

Errors: None.

See also: `Exit` ([1219](#))

Listing: `./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.
```

36.11.126 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 charaters. (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 ([1319](#)), Val ([1336](#)), BinStr ([1191](#))

Listing: ./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.
```

36.11.127 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 from `X`, depending on the size of `X`. If the size of `X` is 4, then the high word is returned. If the size is 2 then the high byte is returned. If the size is 1, the high nibble is returned.

Errors: None

See also: `Lo` ([1252](#))

Listing: `./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.

36.11.128 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` ([1253](#)), `Ord` ([1284](#)), `Pred` ([1288](#)), `Succ` ([1322](#))

Listing: ./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.

36.11.129 HINSTANCE

Synopsis: Windows compatibility type for use in resources

Declaration: `function HINSTANCE : TFPResourceHMODULE`

Visibility: default

Description: This is an opaque type.

36.11.130 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.

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` ([1207](#))

Listing: `./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.

36.11.131 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` ([1217](#))

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` ([1217](#))

36.11.132 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` ([1240](#)), `IndexDWord` ([1241](#)), `IndexWord` ([1242](#)), `CompareByte` ([1198](#))

Listing: `./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.

36.11.133 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 \(1240\)](#), [IndexDWord \(1241\)](#), [IndexWord \(1242\)](#), [CompareChar \(1199\)](#)

Listing: ./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.

36.11.134 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 \(1240\)](#), [IndexChar \(1240\)](#), [IndexWord \(1242\)](#), [IndexDWord \(1241\)](#), [CompareChar0 \(1201\)](#)

36.11.135 IndexDWord

Synopsis: Search for a DWord value in a memory range.

Declaration: function IndexDWord(const buf;len: SizeInt;b: DWord) : SizeInt

Visibility: default

Description: IndexChar 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 \(1240\)](#), [IndexChar \(1240\)](#), [IndexWord \(1242\)](#), [CompareDWord \(1201\)](#)

Listing: `./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.`

36.11.136 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 \(1241\)](#)

36.11.137 Indexword

Synopsis: Search for a WORD value in a memory range.

Declaration: `function Indexword(const buf;len: SizeInt;b: Word) : SizeInt`

Visibility: default

Description: `IndexChar` 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` (1240), `IndexDWord` (1241), `IndexChar` (1240), `CompareWord` (1202)

Listing: `./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.

36.11.138 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` (1214) or `LeaveCriticalSection` (1250) the critical section should be initialized with `InitCriticalSection`.

When a critical section is no longer used, it should be disposed of with `DoneCriticalSection` (1210)

See also: `DoneCriticalSection` (1210), `EnterCriticalSection` (1214), `LeaveCriticalSection` (1250)

36.11.139 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` ([1224](#)), `CopyArray` ([1205](#))

36.11.140 InitThread

Synopsis: Initialize a thread

Declaration: `procedure InitThread(stklen: SizeUInt)`

Visibility: default

Description: Do not use, this is used internally by the thread manager.

36.11.141 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` ([1173](#))

36.11.142 Insert

Synopsis: Insert one string in another.

```
Declaration: procedure Insert(const source: shortstring; var s: shortstring;
                           index: SizeInt)
           procedure Insert(source: Char; var s: shortstring; index: SizeInt)
           procedure Insert(const Source: RawByteString; var S: RawByteString;
                           Index: SizeInt)
           procedure Insert(const Source: UnicodeString; var S: UnicodeString;
                           Index: SizeInt)
           procedure Insert(const Source: WideString; var S: WideString;
                           Index: SizeInt)
```

Visibility: default

Description: `Insert` inserts string `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)

Errors: None.

See also: `Delete` ([1208](#)), `Copy` ([1205](#)), `Pos` ([1287](#))

Listing: `./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.

36.11.143 `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` ([1228](#)), `Round` ([1300](#))

Listing: `./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.

36.11.144 `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` ([1246](#)), `InterLockedIncrement` ([1247](#)), `InterLockedExchange` ([1246](#)), `InterLockedExchangeAdd` ([1247](#))

36.11.145 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` ([1247](#)), `InterLockedExchange` ([1246](#)), `InterLockedExchangeAdd` ([1247](#)), `InterlockedCompareExchange` ([1245](#))

36.11.146 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` ([1246](#)), `InterLockedIncrement` ([1247](#)), `InterLockedExchangeAdd` ([1247](#)), `InterlockedCompareExchange` ([1245](#))

36.11.147 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` ([1246](#)), `InterLockedIncrement` ([1247](#)), `InterLockedExchange` ([1246](#)), `InterlockedCompareExchange` ([1245](#))

36.11.148 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` ([1246](#)), `InterLockedExchange` ([1246](#)), `InterLockedExchangeAdd` ([1247](#)), `InterlockedCompareExchange` ([1245](#))

36.11.149 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 `{\Si-}` 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 :

2File not found.

3Path not found.

4Too many open files.

5Access denied.

- 6Invalid file handle.
- 12Invalid file-access mode.
- 15Invalid disk number.
- 16Cannot remove current directory.
- 17Cannot rename across volumes.

I/O errors :

- 100Error when reading from disk.
- 101Error when writing to disk.
- 102File not assigned.
- 103File not open.
- 104File not opened for input.
- 105File not opened for output.
- 106Invalid number.

Fatal errors :

- 150Disk is write protected.
- 151Unknown device.
- 152Drive not ready.
- 153Unknown command.
- 154CRC check failed.
- 155Invalid drive specified..
- 156Seek error on disk.
- 157Invalid media type.
- 158Sector not found.
- 159Printer out of paper.
- 160Error when writing to device.
- 161Error when reading from device.
- 162Hardware failure.

Errors: None.

Listing: ./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.

36.11.150 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 \(1244\)](#), [FinalizeArray \(1224\)](#), [CopyArray \(1205\)](#), [DynArraySize \(1213\)](#), [DynArrayClear \(1212\)](#), [DynArrayBounds \(1211\)](#), [DynArrayDim \(1212\)](#)

36.11.151 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 \(1309\)](#), [GetMemoryManager \(1231\)](#)

36.11.152 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.

36.11.153 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 it's handle or ID `threadHandle`.

The function returns zero if succesful. A nonzero return value indicates failure.

Errors: If a failure occurred, a nonzero result is returned. The meaning is system dependent.

See also: [WaitForThreadTerminate \(1338\)](#), [EndThread \(1213\)](#), [SuspendThread \(1323\)](#)

36.11.154 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` (1214).

The critical section must have been initialized with `InitCriticalSection` (1243) prior to a call to `EnterCriticalSection` and `LeaveCriticalSection`.

See also: `InitCriticalSection` (1243), `DoneCriticalSection` (1210), `EnterCriticalSection` (1214)

36.11.155 Length

Synopsis: Returns length of a string or array.

Declaration: `function Length(S: AStringType) : Integer`
`function Length(A: DynArrayType) : Integer`

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 dynamical or statical 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` (1287), `SetLength` (1309)

Listing: `./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.

```

36.11.156 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 ([1191](#)), NtoBE ([1257](#)), NtoLE ([1257](#))

36.11.157 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 ([1220](#)), Power ([1288](#))

Listing: ./refex/ex37.pp

Program Example37;

{ Program to demonstrate the Ln function. }

```

begin
  Writeln (Ln(1));      { Prints 0 }
  Writeln (Ln(Exp(1))); { Prints 1 }
end.

```

36.11.158 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 byte of its argument if this is of size 2 (such as `Word` or `SmallInt`), or the low nibble if the size is 1 (such as `byte`). It returns the low word of its argument if this is of type `Longint` or `Cardinal`.

Errors: None.

See also: `Ord` ([1284](#)), `Chr` ([1197](#)), `Hi` ([1236](#))

Listing: `./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.
```

36.11.159 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` ([1229](#)) function.

Errors: On error, 0 is returned.

See also: `FindResource` ([1225](#)), `FreeResource` ([1229](#)), `SizeofResource` ([1316](#)), `LockResource` ([1253](#)), `UnlockResource` ([1333](#)), `FreeResource` ([1229](#))

36.11.160 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` (1333) function.

Errors: if the function fails, `Nil` is returned.

See also: `FindResource` (1225), `FreeResource` (1229), `SizeofResource` (1316), `LoadResource` (1252), `UnlockResource` (1333), `FreeResource` (1229)

36.11.161 longjmp

Synopsis: Jump to address.

Declaration: `procedure longjmp(var S: jmp_buf; value: LongInt)`

Visibility: default

Description: `LongJump` jumps to the address in the `envjmp_buf`, and restores the registers that were stored in it at the corresponding `SetJump` (1308) 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` (1308)

Errors: None.

See also: `SetJump` (1308)

36.11.162 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` (1237).

Errors: None.

See also: High ([1237](#)), Ord ([1284](#)), Pred ([1288](#)), Succ ([1322](#))

36.11.163 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.

Errors: None.

See also: Upcase ([1333](#))

Listing: ./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.
```

36.11.164 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.

36.11.165 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` ([1231](#)), `FreeMem` ([1228](#))

36.11.166 `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` ([1297](#)).

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` ([1197](#)), `Rmdir` ([1297](#))

36.11.167 `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` ([1224](#)), `Fillchar` ([1222](#))

Listing: `./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.

36.11.168 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 \(1255\)](#)

Listing: `./refex/ex109.pp`

Program `Example109;`

{ Program to demonstrate the MoveChar0 function. }

Var

`Buf1, Buf2 : Array[1..80] of char;`
`l : longint;`

begin

`Randomize;`

`For l:=low(buf1) to high(buf1) do`

`Buf1[l]:=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.

36.11.169 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 \(1222\)](#) or [FillWord \(1224\)](#).

For an example, see [Dispose \(1209\)](#).

Errors: What happens if no more memory is available, depends on the value of the variable `ReturnNilIfGrowHeapfails (1182)`: if the variable is `True` then `Nil` is returned. If the variable is `False`, a run-time error is generated.

See also: [Dispose \(1209\)](#), [Freemem \(1228\)](#), [Getmem \(1231\)](#), [ReturnNilIfGrowHeapfails \(1182\)](#)

36.11.170 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` ([1191](#)), `LEtoN` ([1251](#)), `NtoLE` ([1257](#))

36.11.171 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` ([1191](#)), `LEtoN` ([1251](#)), `NtoBE` ([1257](#))

36.11.172 Null

Synopsis: Null variant

Declaration: `function Null : Variant`

Visibility: default

36.11.173 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 ([1319](#)), Val ([1336](#)), BinStr ([1191](#)), HexStr ([1236](#))

Listing: ./refex/ex112.pp

```

Program example112;

  { Program to demonstrate the OctStr function }

  Const Value = 45678;

  Var I : longint;

  begin
    For I:=1 to 10 do
      Writeln ( OctStr(Value,I));
    For I:=1 to 16 do
      Writeln ( OctStr(I,3));
  end.

```

36.11.174 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 ([1184](#)), Ord ([1284](#))

Listing: ./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.

```

36.11.175 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` ([1210](#)), `Cseg` ([1206](#)), `Seg` ([1306](#)), `Ptr` ([1289](#))

Listing: `./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.

36.11.176 operator `*(variant, variant): variant`

Synopsis: Implement multiplication (`*`) operation on variants.

Declaration: `operator operator *(variant, variant): variant(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` ([1261](#))

36.11.177 operator `** (variant, variant): variant`

Synopsis: Implement power (`**`) operation on variants.

Declaration: `operator operator ** (variant, variant): variant(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` ([1259](#))

36.11.178 operator +(variant, variant): variant

Synopsis: Implement addition (+) operation on variants.

Declaration: `operator operator +(variant, variant): variant(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` ([1260](#))

36.11.179 operator -(variant): variant

Synopsis: Implement – (unary minus, negation) operation on variants.

Declaration: `operator operator -(variant): variant(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` ([1260](#))

36.11.180 operator -(variant, variant): variant

Synopsis: Implement subtraction (–) operation on variants.

Declaration: `operator operator -(variant, variant): variant(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` ([1260](#))

36.11.181 operator /(variant, variant): variant

Synopsis: Implement division (/) operation on variants.

Declaration: `operator operator /(variant, variant): variant(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 ([1259](#))

36.11.182 operator :=(ansistring): olevariant

Synopsis:

Declaration: `operator operator :=(ansistring): olevariant(const source: ansistring)
: olevariant`

Visibility: default

Description:

36.11.183 operator :=(ansistring): variant

Synopsis:

Declaration: `operator operator :=(ansistring): variant(const source: ansistring)
: variant`

Visibility: default

Description:

36.11.184 operator :=(Boolean): olevariant

Synopsis:

Declaration: `operator operator :=(Boolean): olevariant(const source: Boolean)
: olevariant`

Visibility: default

Description:

36.11.185 operator :=(Boolean): variant

Synopsis:

Declaration: `operator operator :=(Boolean): variant(const source: Boolean) : variant`

Visibility: default

Description:

36.11.186 operator :=(Byte): olevariant

Synopsis:

Declaration: `operator operator :=(Byte): olevariant(const source: Byte) : olevariant`

Visibility: default

Description:

36.11.187 operator :=(Byte): variant

Synopsis:

Declaration: `operator operator :=(Byte): variant(const source: Byte) : variant`

Visibility: default

Description:

36.11.188 operator :=(Char): olevariant

Synopsis:

Declaration: `operator operator :=(Char): olevariant(const source: Char) : olevariant`

Visibility: default

Description:

36.11.189 operator :=(Char): variant

Synopsis:

Declaration: `operator operator :=(Char): variant(const source: Char) : variant`

Visibility: default

Description:

36.11.190 operator :=(comp): olevariant

Declaration: `operator operator :=(comp): olevariant(const source: comp) : olevariant`

Visibility: default

36.11.191 operator :=(comp): variant

Declaration: `operator operator :=(comp): variant(const source: comp) : variant`

Visibility: default

36.11.192 operator :=(currency): olevariant

Synopsis:

Declaration: `operator operator :=(currency): olevariant(const source: currency)
: olevariant`

Visibility: default

Description:

36.11.193 operator :=(currency): variant

Synopsis:

Declaration: `operator operator :=(currency): variant(const source: currency)
: variant`

Visibility: default

Description:

36.11.194 operator :=(Double): olevariant

Synopsis:

Declaration: `operator operator :=(Double): olevariant(const source: Double)
: olevariant`

Visibility: default

Description:

36.11.195 operator :=(Double): variant

Synopsis:

Declaration: `operator operator :=(Double): variant(const source: Double) : variant`

Visibility: default

Description:

36.11.196 operator :=(DWord): olevariant

Synopsis:

Declaration: `operator operator :=(DWord): olevariant(const source: DWord)
: olevariant`

Visibility: default

Description:

36.11.197 operator :=(DWord): variant

Synopsis:

Declaration: `operator operator :=(DWord): variant(const source: DWord) : variant`

Visibility: default

Description:

36.11.198 operator :=(extended): olevariant

Declaration: `operator operator :=(extended): olevariant(const source: extended)
: olevariant`

Visibility: default

36.11.199 operator :=(extended): variant

Declaration: `operator operator :=(extended): variant(const source: extended)
: variant`

Visibility: default

36.11.200 operator :=(Int64): olevariant

Synopsis:

Declaration: `operator operator :=(Int64): olevariant(const source: Int64)
: olevariant`

Visibility: default

Description:

36.11.201 operator :=(Int64): variant

Synopsis:

Declaration: `operator operator :=(Int64): variant(const source: Int64) : variant`

Visibility: default

Description:

36.11.202 operator :=(longbool): olevariant

Synopsis:

Declaration: `operator operator :=(longbool): olevariant(const source: longbool)
: olevariant`

Visibility: default

Description:

36.11.203 operator :=(longbool): variant

Synopsis:

Declaration: `operator operator :=(longbool): variant(const source: longbool)
: variant`

Visibility: default

Description:

36.11.204 operator :=(LongInt): olevariant

Synopsis:

Declaration: `operator operator :=(LongInt): olevariant(const source: LongInt)
: olevariant`

Visibility: default

Description:

36.11.205 operator :=(LongInt): variant

Synopsis:

Declaration: `operator operator :=(LongInt): variant(const source: LongInt) : variant`

Visibility: default

Description:

36.11.206 operator :=(olevariant): ansistring

Synopsis:

Declaration: `operator operator :=(olevariant): ansistring(const source: olevariant)
: ansistring`

Visibility: default

Description:

36.11.207 operator :=(olevariant): Boolean

Synopsis:

Declaration: `operator operator :=(olevariant): Boolean(const source: olevariant)
: Boolean`

Visibility: default

Description:

36.11.208 operator :=(olevariant): Byte

Synopsis:

Declaration: `operator operator :=(olevariant): Byte(const source: olevariant) : Byte`

Visibility: default

Description:

36.11.209 operator :=(olevariant): Char

Synopsis:

Declaration: `operator operator :=(olevariant): Char(const source: olevariant) : Char`

Visibility: default

Description:

36.11.210 operator :=(olevariant): comp

Declaration: `operator operator :=(olevariant): comp(const source: olevariant) : comp`

Visibility: default

36.11.211 operator :=(olevariant): currency

Synopsis:

Declaration: `operator operator :=(olevariant): currency(const source: olevariant)
: currency`

Visibility: default

Description:

36.11.212 operator :=(olevariant): Double

Synopsis:

Declaration: `operator operator :=(olevariant): Double(const source: olevariant)
: Double`

Visibility: default

Description:

36.11.213 operator :=(olevariant): DWord

Synopsis:

Declaration: `operator operator :=(olevariant): DWord(const source: olevariant)
: DWord`

Visibility: default

Description:

36.11.214 operator :=(olevariant): extended

Declaration: `operator operator :=(olevariant): extended(const source: olevariant)
: extended`

Visibility: default

36.11.215 operator :=(olevariant): Int64

Synopsis:

Declaration: `operator operator :=(olevariant): Int64(const source: olevariant)
: Int64`

Visibility: default

Description:

36.11.216 operator :=(olevariant): longbool

Synopsis:

Declaration: `operator operator :=(olevariant): longbool(const source: olevariant)
: longbool`

Visibility: default

Description:

36.11.217 operator :=(olevariant): LongInt

Synopsis:

Declaration: `operator operator :=(olevariant): LongInt(const source: olevariant)
: LongInt`

Visibility: default

Description:

36.11.218 operator :=(olevariant): QWord

Synopsis:

Declaration: `operator operator :=(olevariant): QWord(const source: olevariant)
: QWord`

Visibility: default

Description:

36.11.219 operator :=(olevariant): Real

Declaration: `operator operator :=(olevariant): Real(const source: olevariant) : Real`

Visibility: default

36.11.220 operator :=(olevariant): ShortInt

Synopsis:

Declaration: `operator operator :=(olevariant): ShortInt(const source: olevariant)
: ShortInt`

Visibility: default

Description:

36.11.221 operator :=(olevariant): shortstring

Synopsis:

Declaration: `operator operator :=(olevariant): shortstring(const source: olevariant)
: shortstring`

Visibility: default

Description:

36.11.222 operator :=(olevariant): single

Declaration: `operator operator :=(olevariant): single(const source: olevariant)
: single`

Visibility: default

36.11.223 operator :=(olevariant): SmallInt

Synopsis:

Declaration: `operator operator :=(olevariant): SmallInt(const source: olevariant)
: SmallInt`

Visibility: default

Description:

36.11.224 operator :=(olevariant): TDateTime

Synopsis:

Declaration: `operator operator :=(olevariant): TDateTime(const source: olevariant)
: TDateTime`

Visibility: default

Description:

36.11.225 operator :=(olevariant): TError

Synopsis:

Declaration: `operator operator :=(olevariant): TError(const source: olevariant)
: TError`

Visibility: default

Description:

36.11.226 operator :=(olevariant): UnicodeString

Declaration: `operator operator :=(olevariant): UnicodeString
(const source: olevariant)
: UnicodeString`

Visibility: default

36.11.227 operator :=(olevariant): variant

Synopsis:

Declaration: `operator operator :=(olevariant): variant(const source: olevariant)
: variant`

Visibility: default

Description:

36.11.228 operator :=(olevariant): WideChar

Synopsis:

Declaration: `operator operator :=(olevariant): WideChar(const source: olevariant)
: WideChar`

Visibility: default

Description:

36.11.229 operator :=(olevariant): widestring

Synopsis:

Declaration: `operator operator :=(olevariant): widestring(const source: olevariant)
: widestring`

Visibility: default

Description:

36.11.230 operator :=(olevariant): Word

Synopsis:

Declaration: `operator operator :=(olevariant): Word(const source: olevariant) : Word`

Visibility: default

Description:

36.11.231 operator :=(olevariant): wordbool

Synopsis:

Declaration: `operator operator :=(olevariant): wordbool(const source: olevariant)
: wordbool`

Visibility: default

Description:

36.11.232 operator :=(QWord): olevariant

Synopsis:

Declaration: `operator operator :=(QWord): olevariant(const source: QWord)
: olevariant`

Visibility: default

Description:

36.11.233 operator :=(QWord): variant

Synopsis:

Declaration: `operator operator :=(QWord): variant(const source: QWord) : variant`

Visibility: default

Description:

36.11.234 operator :=(Real): olevariant

Declaration: `operator operator :=(Real): olevariant(const source: Real) : olevariant`

Visibility: default

36.11.235 operator :=(Real): variant

Declaration: `operator operator :=(Real): variant(const source: Real) : variant`

Visibility: default

36.11.236 operator :=(real48): Double

Synopsis:

Declaration: `operator operator :=(real48) : Double(b: real48) : Double`

Visibility: default

Description:

36.11.237 operator :=(real48): extended

Declaration: `operator operator :=(real48) : extended(b: real48) : extended`

Visibility: default

36.11.238 operator :=(ShortInt): olevariant

Synopsis:

Declaration: `operator operator :=(ShortInt) : olevariant(const source: ShortInt)
: olevariant`

Visibility: default

Description:

36.11.239 operator :=(ShortInt): variant

Synopsis:

Declaration: `operator operator :=(ShortInt) : variant(const source: ShortInt)
: variant`

Visibility: default

Description:

36.11.240 operator :=(shortstring): olevariant

Synopsis:

Declaration: `operator operator :=(shortstring) : olevariant(const source: shortstring)
: olevariant`

Visibility: default

Description:

36.11.241 operator :=(shortstring): variant

Synopsis:

Declaration: `operator operator :=(shortstring) : variant(const source: shortstring)
: variant`

Visibility: default

Description:

36.11.242 operator :=(single): olevariant

Declaration: `operator operator :=(single): olevariant(const source: single)
: olevariant`

Visibility: default

36.11.243 operator :=(single): variant

Declaration: `operator operator :=(single): variant(const source: single) : variant`

Visibility: default

36.11.244 operator :=(SmallInt): olevariant

Synopsis:

Declaration: `operator operator :=(SmallInt): olevariant(const source: SmallInt)
: olevariant`

Visibility: default

Description:

36.11.245 operator :=(SmallInt): variant

Synopsis:

Declaration: `operator operator :=(SmallInt): variant(const source: SmallInt)
: variant`

Visibility: default

Description:

36.11.246 operator :=(TDateTime): olevariant

Synopsis:

Declaration: `operator operator :=(TDateTime): olevariant(const source: TDateTime)
: olevariant`

Visibility: default

Description:

36.11.247 operator :=(TDateTime): variant

Synopsis:

Declaration: `operator operator :=(TDateTime): variant(const source: TDateTime)
: variant`

Visibility: default

Description:

36.11.248 operator :=(TError): olevariant

Synopsis:

Declaration: `operator operator :=(TError): olevariant(const source: TError)
: olevariant`

Visibility: default

Description:

36.11.249 operator :=(TError): variant

Synopsis:

Declaration: `operator operator :=(TError): variant(const source: TError) : variant`

Visibility: default

Description:

36.11.250 operator :=(UCS4String): variant

Declaration: `operator operator :=(UCS4String): variant(const source: UCS4String)
: variant`

Visibility: default

36.11.251 operator :=(UnicodeString): olevariant

Declaration: `operator operator :=(UnicodeString): olevariant
(const source: UnicodeString)
: olevariant`

Visibility: default

36.11.252 operator :=(UnicodeString): variant

Declaration: `operator operator :=(UnicodeString): variant
(const source: UnicodeString)
: variant`

Visibility: default

36.11.253 operator :=(UTF8String): variant

Declaration: `operator operator :=(UTF8String): variant(const source: UTF8String)
: variant`

Visibility: default

36.11.254 operator :=(variant): ansistring

Synopsis:

Declaration: `operator operator :=(variant): ansistring(const source: variant)
: ansistring`

Visibility: default

Description:

36.11.255 operator :=(variant): Boolean

Synopsis:

Declaration: `operator operator :=(variant): Boolean(const source: variant) : Boolean`

Visibility: default

Description:

36.11.256 operator :=(variant): Byte

Synopsis:

Declaration: `operator operator :=(variant): Byte(const source: variant) : Byte`

Visibility: default

Description:

36.11.257 operator :=(variant): Char

Synopsis:

Declaration: `operator operator :=(variant): Char(const source: variant) : Char`

Visibility: default

Description:

36.11.258 operator :=(variant): comp

Declaration: `operator operator :=(variant): comp(const source: variant) : comp`

Visibility: default

36.11.259 operator :=(variant): currency

Synopsis:

Declaration: `operator operator :=(variant): currency(const source: variant)
: currency`

Visibility: default

Description:

36.11.260 operator :=(variant): Double

Synopsis:

Declaration: `operator operator :=(variant): Double(const source: variant) : Double`

Visibility: default

Description:

36.11.261 operator :=(variant): DWord

Synopsis:

Declaration: `operator operator :=(variant): DWord(const source: variant) : DWord`

Visibility: default

Description:

36.11.262 operator :=(variant): extended

Declaration: `operator operator :=(variant): extended(const source: variant)
: extended`

Visibility: default

36.11.263 operator :=(variant): Int64

Synopsis:

Declaration: `operator operator :=(variant): Int64(const source: variant) : Int64`

Visibility: default

Description:

36.11.264 operator :=(variant): longbool

Synopsis:

Declaration: `operator operator :=(variant): longbool(const source: variant)
: longbool`

Visibility: default

Description:

36.11.265 operator :=(variant): LongInt

Synopsis:

Declaration: `operator operator :=(variant): LongInt(const source: variant) : LongInt`

Visibility: default

Description:

36.11.266 operator :=(variant): olevariant

Synopsis:

Declaration: `operator operator :=(variant): olevariant(const source: variant)
: olevariant`

Visibility: default

Description:

36.11.267 operator :=(variant): QWord

Synopsis:

Declaration: `operator operator :=(variant): QWord(const source: variant) : QWord`

Visibility: default

Description:

36.11.268 operator :=(variant): Real

Declaration: `operator operator :=(variant): Real(const source: variant) : Real`

Visibility: default

36.11.269 operator :=(variant): ShortInt

Synopsis:

Declaration: `operator operator :=(variant): ShortInt(const source: variant)
: ShortInt`

Visibility: default

Description:

36.11.270 operator :=(variant): shortstring

Synopsis:

Declaration: `operator operator :=(variant): shortstring(const source: variant)
: shortstring`

Visibility: default

Description:

36.11.271 operator :=(variant): single

Declaration: `operator operator :=(variant): single(const source: variant) : single`

Visibility: default

36.11.272 operator :=(variant): SmallInt

Synopsis:

Declaration: `operator operator :=(variant): SmallInt(const source: variant)
: SmallInt`

Visibility: default

Description:

36.11.273 operator :=(variant): TDateTime

Synopsis:

Declaration: `operator operator :=(variant): TDateTime(const source: variant)
: TDateTime`

Visibility: default

Description:

36.11.274 operator :=(variant): TError

Synopsis:

Declaration: `operator operator :=(variant): TError(const source: variant) : TError`

Visibility: default

Description:

36.11.275 operator :=(variant): unicodestring

Declaration: `operator operator :=(variant): unicodestring(const source: variant)
: unicodestring`

Visibility: default

36.11.276 operator :=(variant): UTF8String

Declaration: `operator operator :=(variant): UTF8String(const source: variant)
: UTF8String`

Visibility: default

36.11.277 operator :=(variant): WideChar

Synopsis:

Declaration: `operator operator :=(variant): WideChar(const source: variant)
: WideChar`

Visibility: default

Description:

36.11.278 operator :=(variant): widestring

Synopsis:

Declaration: `operator operator :=(variant): widestring(const source: variant)
: widestring`

Visibility: default

Description:

36.11.279 operator :=(variant): Word

Synopsis:

Declaration: `operator operator :=(variant): Word(const source: variant) : Word`

Visibility: default

Description:

36.11.280 operator :=(variant): wordbool

Synopsis:

Declaration: `operator operator :=(variant): wordbool(const source: variant)
: wordbool`

Visibility: default

Description:

36.11.281 operator :=(WideChar): olevariant

Synopsis:

Declaration: `operator operator :=(WideChar): olevariant(const source: WideChar)
: olevariant`

Visibility: default

Description:

36.11.282 operator :=(WideChar): variant

Synopsis:

Declaration: `operator operator :=(WideChar): variant(const source: WideChar)
: variant`

Visibility: default

Description:

36.11.283 operator :=(widestring): olevariant

Synopsis:

Declaration: `operator operator :=(widestring): olevariant(const source: widestring)
: olevariant`

Visibility: default

Description:

36.11.284 operator :=(widestring): variant

Synopsis:

Declaration: `operator operator :=(widestring): variant(const source: widestring)
: variant`

Visibility: default

Description:

36.11.285 operator :=(Word): olevariant

Synopsis:

Declaration: `operator operator :=(Word): olevariant(const source: Word) : olevariant`

Visibility: default

Description:

36.11.286 operator :=(Word): variant

Synopsis:

Declaration: `operator operator :=(Word): variant(const source: Word) : variant`

Visibility: default

Description:

36.11.287 operator :=(wordbool): olevariant

Synopsis:

Declaration: `operator operator :=(wordbool): olevariant(const source: wordbool)
: olevariant`

Visibility: default

Description:

36.11.288 operator :=(wordbool): variant

Synopsis:

Declaration: `operator operator :=(wordbool): variant(const source: wordbool)
: variant`

Visibility: default

Description:

36.11.289 operator <(variant, variant): Boolean

Synopsis: Implement < (less than) operation on variants.

Declaration: `operator operator <(variant, variant): Boolean(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 `opcmplt`.

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 ([1281](#))

36.11.290 operator <=(variant, variant): Boolean

Synopsis: Implement <= (less than or equal) operation on variants.

Declaration: `operator operator <=(variant, variant): Boolean(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 `opcmple`.

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 ([1280](#))

36.11.291 operator =(variant, variant): Boolean

Synopsis: Implement = (equality) operation on variants.

Declaration: `operator operator =(variant, variant): Boolean(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 ([1280](#))

36.11.292 operator >(variant, variant): Boolean

Synopsis: Implement > (greater than) operation on variants.

Declaration: `operator operator >(variant, variant): Boolean(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 ([1280](#))

36.11.293 operator >=(variant, variant): Boolean

Synopsis: Implement >= (greater than or equal) operation on variants.

Declaration: `operator operator >=(variant, variant): Boolean(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 ([1280](#))

36.11.294 operator and(variant, variant): variant

Synopsis: Implement logical/binary and operation on variants

Declaration: `operator operator and(variant, variant): variant(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 ([1283](#)), operator xor(variant, variant): variant ([1284](#)), operator not(variant): variant ([1282](#))

36.11.295 operator div(variant, variant): variant

Synopsis: Implement `div` (integer division) operation on variants.

Declaration: `operator operator div(variant, variant): variant(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` ([1282](#))

36.11.296 operator mod(variant, variant): variant

Synopsis: Implement `mod` (modulo) operation on variants.

Declaration: `operator operator mod(variant, variant): variant(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` ([1282](#))

36.11.297 operator not(variant): variant

Synopsis: Implement logical/binary `not` operation on variants

Declaration: `operator operator not(variant): variant(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` ([1281](#)), `operator or(variant, variant): variant` ([1283](#)), `operator xor(variant, variant): variant` ([1284](#))

36.11.298 operator or(variant, variant): variant

Synopsis: Implement logical/binary `or` operation on variants

Declaration: `operator operator or(variant, variant): variant(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` ([1281](#)), `operator xor(variant, variant): variant` ([1284](#)), `operator not(variant): variant` ([1282](#))

36.11.299 operator shl(variant, variant): variant

Synopsis: Implement binary `shl` operation on variants.

Declaration: `operator operator shl(variant, variant): variant(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` ([1283](#))

36.11.300 operator shr(variant, variant): variant

Synopsis: Implement binary `shr` operation on variants.

Declaration: `operator operator shr(variant, variant): variant(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` ([1283](#))

36.11.301 operator xor(variant, variant): variant

Synopsis: Implement logical/binary `xor` operation on variants

Declaration: `operator operator xor(variant, variant): variant(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 ([1283](#)), operator `and`(variant, variant): variant ([1281](#)), operator `not`(variant): variant ([1282](#))

36.11.302 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` ([1197](#)), `Succ` ([1322](#)), `Pred` ([1288](#)), `High` ([1237](#)), `Low` ([1253](#))

Listing: `./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.

36.11.303 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` ([1333](#))

36.11.304 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` ([1285](#))

Listing: `./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.
```

36.11.305 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 return 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)

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` ([1285](#)).

Errors: None.

See also: `Paramcount` ([1285](#))

36.11.306 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` ([1206](#)), `Sin` ([1315](#))

Listing: `./refex/ex47.pp`

```
Program Example47;

{ Program to demonstrate the Pi function. }

begin
  Writeln ( Pi );           { 3.1415926 }
  Writeln ( Sin( Pi ) );
end.
```

36.11.307 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`.

36.11.308 Pos

Synopsis: Search for substring in a string.

Declaration:

```

function Pos(const substr: shortstring;const s: shortstring) : SizeInt
function Pos(C: Char;const s: shortstring) : SizeInt
function Pos(const Substr: ShortString;const Source: RawByteString)
    : SizeInt
function pos(const substr: shortstring;c: Char) : SizeInt
function Pos(const Substr: RawByteString;const Source: RawByteString)
    : SizeInt
function Pos(c: AnsiChar;const s: RawByteString) : SizeInt
function Pos(const Substr: UnicodeString;const Source: UnicodeString)
    : SizeInt
function Pos(c: Char;const s: UnicodeString) : SizeInt
function Pos(c: UnicodeChar;const s: UnicodeString) : SizeInt
function Pos(const c: RawByteString;const s: UnicodeString) : SizeInt
function Pos(const c: UnicodeString;const s: RawByteString) : SizeInt
function Pos(const c: ShortString;const s: UnicodeString) : SizeInt
function Pos(const Substr: WideString;const Source: WideString)
    : SizeInt
function Pos(c: Char;const s: WideString) : SizeInt
function Pos(c: WideChar;const s: WideString) : SizeInt
function Pos(c: WideChar;const s: RawByteString) : SizeInt
function Pos(const c: RawByteString;const s: WideString) : SizeInt
function Pos(const c: WideString;const s: RawByteString) : SizeInt
function Pos(const c: ShortString;const s: WideString) : 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 in S, if S contains Substr. In case Substr isn't found, 0 is returned. The search is case-sensitive.

Errors: None

See also: Length ([1250](#)), Copy ([1205](#)), Delete ([1208](#)), Insert ([1244](#))

Listing: ./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.
```

36.11.309 Power

Synopsis: Raise float to integer power

Declaration: `function Power(Base: Double;expon: Double) : Double`
`function Power(Base: LongInt;expon: LongInt) : LongInt`

Visibility: default

Description: `Power` returns the value of `base` to the power `expon`. `Base` and `expon` can be of type `Longint`, in which case the result will also be a `Longint`.

The function actually returns `Exp (expon*Ln (base))`

Errors: None.

See also: `Exp` ([1220](#)), `Ln` ([1251](#))

Listing: `./refex/ex78.pp`

Program `Example78;`

{ Program to demonstrate the Power function. }

```
begin
  Writeln (Power(exp(1.0),1.0):8:2); { Should print 2.72 }
end.
```

36.11.310 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` ([1284](#))

Errors: Run-time error 201 is generated when the result is out of range.

See also: `Ord` ([1284](#)), `Succ` ([1322](#)), `High` ([1237](#)), `Low` ([1253](#))

36.11.311 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.

36.11.312 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` ([1185](#))

Listing: `./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.

36.11.313 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).

36.11.314 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`. 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 the Mersenne Twister 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.

Errors: None.

See also: `Randomize` ([1290](#))

Listing: `./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.

36.11.315 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` ([1290](#)).

Errors: None.

See also: [Random \(1290\)](#)

36.11.316 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 \(1292\)](#), [Blockread \(1192\)](#), [Write \(1340\)](#), [Blockwrite \(1193\)](#)

Listing: ./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.
```

36.11.317 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` ([1292](#)), `ReadWriteBarrier` ([1293](#)), `WriteBarrier` ([1340](#))

36.11.318 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` ([1291](#)), `ReadWriteBarrier` ([1293](#)), `WriteBarrier` ([1340](#))

36.11.319 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` ([1291](#)).

Errors: If no data is available, empty values are returned (0 for ordinal values, empty strings for string values)

See also: `Read` ([1291](#)), `Blockread` ([1192](#)), `Write` ([1340](#)), `Blockwrite` ([1193](#))

36.11.320 ReadStr

Synopsis: Read variables from a string

Declaration: `procedure ReadStr(const S: string; Args: Arguments)`

Visibility: `default`

Description: `ReadStr` behaves like `Read` ([1291](#)), 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` ([1291](#)).

See also: `Read` ([1291](#)), `WriteStr` ([1341](#)), `Write` ([1340](#))

36.11.321 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` ([1291](#)), `ReadDependencyBarrier` ([1292](#)), `WriteBarrier` ([1340](#))

36.11.322 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: `./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.

```

36.11.323 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` ([1231](#)), `FreeMem` ([1228](#))

36.11.324 ReAllocMemory

Synopsis: Alias for `ReAllocMem` ([1294](#))

Declaration: `function ReAllocMemory(p: pointer;Size: PtrUInt) : pointer`

Visibility: default

Description: `ReAllocMemory` is an alias for `ReAllocMem` ([1294](#)).

See also: `ReAllocMem` ([1294](#))

36.11.325 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` (1184) 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` (1184)

36.11.326 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` (1217)

Listing: `./refex/ex77.pp`

Program `Example77`;

```

{ Program to demonstrate the Rename function. }
Var F : Text;

begin
  Assign (F, paramstr(1));
  Rename (F, paramstr(2));
end.

```

36.11.327 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. 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.`

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` ([1296](#)), `Assign` ([1188](#)), `Close` ([1198](#)), `Append` ([1186](#))

Listing: `./refex/ex51.pp`

Program `Example51` ;

{ Program to demonstrate the Reset function. }

Function `FileExists (Name : String) : boolean`;

Var `F : File`;

begin

{SI-}

`Assign (F,Name);`

`Reset (F);`

{SI+}

`FileExists := (IOResult=0) and (Name<>'');`

`Close (f);`

end;

begin

`If FileExists (Paramstr(1)) then`

`Writeln ('File found')`

`else`

`Writeln ('File NOT found');`

end.

36.11.328 ResumeThread

Synopsis: Resume a suspended thread.

Declaration: `function ResumeThread(threadHandle: TThreadID) : DWord`

Visibility: default

Description: `ResumeThread` causes a suspended thread (using `SuspendThread` ([1323](#))) to resume its execution. 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: `SuspendThread` ([1323](#)), `KillThread` ([1249](#))

36.11.329 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` (1295) is needed. File sharing is not taken into account when calling `Rewrite`.

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` (1295), `Assign` (1188), `Close` (1198), `Flush` (1226), `Append` (1186)

Listing: `./refex/ex52.pp`

Program Example52;

{ Program to demonstrate the Rewrite function. }

Var `F : File`;
`l : longint`;

begin
`Assign (F, 'Test.tmp');`
{ Create the file. Recordsize is 4 }
`Rewrite (F, Sizeof(l));`
`For l:=1 to 10 do`
`BlockWrite (F,l,1);`
`close (f);`
{ F contains now a binary representation of
10 longints going from 1 to 10 }
end.

36.11.330 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` (1197), `Mkdir` (1255)

Listing: `./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.

```

36.11.331 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 ([1299](#)), RolWord ([1299](#)), RolDWord ([1298](#)), RolQWord ([1299](#))

36.11.332 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 ([1298](#)), RolWord ([1299](#)), RorDWord ([1299](#)), RolQWord ([1299](#))

36.11.333 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` ([1298](#)), `RolWord` ([1299](#)), `RolDWord` ([1298](#)), `RorQWord` ([1300](#))

36.11.334 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` ([1298](#)), `RorWord` ([1300](#)), `RolDWord` ([1298](#)), `RolQWord` ([1299](#))

36.11.335 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` ([1298](#)), `RorWord` ([1300](#)), `RorDWord` ([1299](#)), `RorQWord` ([1300](#))

36.11.336 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 \(1299\)](#), [RolDWord \(1298\)](#), [RorWord \(1300\)](#), [RorQWord \(1300\)](#)

36.11.337 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 \(1299\)](#), [RorWord \(1300\)](#), [RorDWord \(1299\)](#), [RolQWord \(1299\)](#)

36.11.338 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 \(1299\)](#), [RolWord \(1299\)](#), [RorDWord \(1299\)](#), [RorQWord \(1300\)](#)

36.11.339 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: None.

See also: [Frac \(1228\)](#), [Int \(1245\)](#), [Trunc \(1329\)](#)

Listing: `./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.
```

36.11.340 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` ([1301](#))

`RTLEvent` is used mainly for the `synchronize` method.

See also: `RTLEventDestroy` ([1301](#)), `RTLEventSetEvent` ([1302](#)), `RTLEventReSetEvent` ([1301](#)), `RTLEventWaitFor` ([1302](#))

36.11.341 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` ([1301](#)), `RTLEventResetEvent` ([1301](#)), `RTLEventSetEvent` ([1302](#))

36.11.342 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 \(1301\)](#), [RTLEventDestroy \(1301\)](#), [RTLEventSetEvent \(1302\)](#), [RTLEventWaitFor \(1302\)](#)

36.11.343 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 \(1301\)](#), [RTLEventResetEvent \(1301\)](#), [RTLEventDestroy \(1301\)](#), [RTLEventWaitFor \(1302\)](#)

36.11.344 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 (1302)` 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 \(1301\)](#), [RTLEventDestroy \(1301\)](#), [RTLEventSetEvent \(1302\)](#), [RTLEventWaitFor \(1302\)](#)

36.11.345 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 \(1219\)](#), [Halt \(1235\)](#)

Listing: `./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.
```

36.11.346 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` ([1303](#)), `SarSmallInt` ([1303](#)), `SarLongInt` ([1303](#))

36.11.347 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` ([1303](#)), `SarSmallInt` ([1303](#)), `SarInt64` ([1303](#))

36.11.348 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` ([1303](#)), `SarLongint` ([1303](#)), `SarInt64` ([1303](#))

36.11.349 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` ([1303](#)), `SarLongint` ([1303](#)), `SarInt64` ([1303](#))

36.11.350 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` (1295) or `Rewrite` (1296), 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` (1215), `SeekEof` (1304), `SeekEoln` (1305)

Listing: `./refex/ex56.pp`

Program `Example56`;

{ Program to demonstrate the Seek function. }

Var

`F : File;`
`I, 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 beginning 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;

`Close (f);`

end.

36.11.351 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` if 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` ([1215](#)), `SeekEoln` ([1305](#)), `Seek` ([1304](#))

Listing: `./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.
```

36.11.352 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` ([1215](#)), `SeekEof` ([1304](#)), `Seek` ([1304](#))

Listing: `./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.

```

36.11.353 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` ([1210](#)), `CSeg` ([1206](#)), `Ofs` ([1258](#)), `Ptr` ([1289](#))

Listing: `./refex/ex60.pp`

Program `Example60`;

```

{ Program to demonstrate the Seg function. }
Var
  W : Word;

begin
  W:=Seg(W); { W contains its own Segment}
end.

```

36.11.354 SemaphoreDestroy

Synopsis: Destroy a semaphore

Declaration: `procedure SemaphoreDestroy(const sem: Pointer)`

Visibility: default

Description: `SemaphoreDestroy` destroys a semaphore, created with `SemaphoreInit` ([1307](#)).

The use of semaphores requires thread support (use `#rtl.threads` ([446](#)) unit on unices).

Errors: If no thread support is compiled in, then calling this function will result in a run-time error 232.

See also: `#rtl.threads` ([446](#)), `SemaphoreWait` ([1307](#)), `SemaphorePost` ([1307](#)), `SemaphoreInit` ([1307](#))

36.11.355 SemaphoreInit

Synopsis: Create a new semaphore

Declaration: `function SemaphoreInit : Pointer`

Visibility: default

Description: `SemaphoreInit` creates a new semaphore, which can be used in thread synchronization. The semaphore can be used with `SemaphoreWait` (1307) and `SemaphorePost` (1307), and must be destroyed with `SemaphoreDestroy` (1306).

The result is an untyped pointer (thread support is platform dependent).

The use of semaphores requires thread support (use `#rtl.threads` (446) unit on unices).

Errors: On error, `Pointer(-1)` is returned. If no thread support is compiled in, then calling this function will result in a run-time error 232.

See also: `#rtl.threads` (446), `SemaphoreWait` (1307), `SemaphorePost` (1307), `SemaphoreDestroy` (1306)

36.11.356 SemaphorePost

Synopsis: Make semaphore available

Declaration: `procedure SemaphorePost(const sem: Pointer)`

Visibility: default

Description: `SemaphorePost` makes the semaphore `FSem` to available. Any threads that were waiting for the semaphore `FSem` to become available (using `SemaphoreWait` (1307), will continue to execute.

The use of semaphores requires thread support (use `#rtl.threads` (446) unit on unices).

Errors: If no thread support is compiled in, then calling this function will result in a run-time error 232.

See also: `#rtl.threads` (446), `SemaphoreInit` (1307), `SemaphoreWait` (1307), `SemaphoreDestroy` (1306)

36.11.357 SemaphoreWait

Synopsis: Wait for semaphore to become available

Declaration: `procedure SemaphoreWait(const sem: Pointer)`

Visibility: default

Description: `SemaphoreWait` waits (indefinitely) for the semaphore `FSem` to become available.

The use of semaphores requires thread support (use `#rtl.threads` (446) unit on unices).

Errors: If no thread support is compiled in, then calling this function will result in a run-time error 232.

See also: `#rtl.threads` (446), `SemaphoreInit` (1307), `SemaphorePost` (1307), `SemaphoreDestroy` (1306)

36.11.358 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` ([1319](#))

36.11.359 Setjmp

Synopsis: Save current execution point.

Declaration: `function Setjmp (var S: jmp_buf) : LongInt`

Visibility: default

Description: `SetJmp` fills `env` 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 `LongJump` ([1253](#)) with `env` as an argument was made somewhere in the program.

Errors: None.

See also: `LongJump` ([1253](#))

Listing: `./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. ');
            dojmp (env, 2);
        end
    else
        writeln ('Passed second time. ');
end.
```

36.11.360 SetLength

Synopsis: Set length of a string.

Declaration: `procedure SetLength(var S: AStringType; Len: Integer)`
`procedure SetLength(var A: DynArrayType; Len: Integer)`

Visibility: default

Description: `SetLength` 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` *\emph{must}* be used to set the length of the string.

In the case of a dynamical 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.

Errors: None.

See also: `Length` ([1250](#))

Listing: `./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.

36.11.361 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` ([1231](#)), `IsMemoryManagerSet` ([1249](#))

36.11.362 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` (1181) 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.

See also: `DefaultSystemCodePage` (1181), `SetMultiByteFileSystemCodePage` (1310), `SetMultiByteRTLFileSystemCodePage` (1310)

36.11.363 `SetMultiByteFileSystemCodePage`

Synopsis: Set codepage used when passing strings to OS single-byte filesystem APIs

Declaration: `procedure SetMultiByteFileSystemCodePage (CodePage: TSystemCodePage)`

Visibility: default

Description: `SetMultiByteFileSystemCodePage` sets the codepage used in single-byte OS filesystem 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 Filesystem APIs changes, and strings passed to the codepage-aware filesystem APIs will be passed using the new codepage.

This constant is not used if the filesystem 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` (1180), `SetMultiByteConversionCodePage` (1309), `SetMultiByteRTLFileSystemCodePage` (1310)

36.11.364 `SetMultiByteRTLFileSystemCodePage`

Synopsis: Set codepage used when interpreting strings from OS single-byte filesystem APIs

Declaration: `procedure SetMultiByteRTLFileSystemCodePage (CodePage: TSystemCodePage)`

Visibility: default

Description: `SetMultiByteRTLFileSystemCodePage` sets the codepage used to interpret strings returned by single-byte OS filesystem 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 filesystem APIs will be interpreted using the new codepage.

his constant is not used if the filesystem 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` (1310), `SetMultiByteConversionCodePage` (1309), `SetMultiByteRTLFileSystemCodePage` (1310)

36.11.365 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` ([1169](#)), `GetResourceManager` ([1232](#))

36.11.366 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` ([1309](#))

36.11.367 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 (1296), Reset (1295) 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 (1188), Reset (1295), Rewrite (1296), Append (1186)

Listing: ./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.

36.11.368 SetTextCodePage

Synopsis: Set the codepage used in a text file.

Declaration: procedure SetTextCodePage (var T: Text; CodePage: TSystemCodePage)

Visibility: default

Description: GetTextCodePage 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 (1164), GetTextCodePage (1232)

36.11.369 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` (1125).

Errors: None.

See also: `DefaultTextLineBreakStyle` (1125), `TTextLineBreakStyle` (1172)

36.11.370 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 succesfully, `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` (1233), `TThreadManager` (1173)

36.11.371 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` (1174)

36.11.372 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` ([1233](#))

36.11.373 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` ([1178](#))

36.11.374 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` ([1408](#))

36.11.375 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^{63} , then the result is undefined.

Errors: None.

See also: `Cos` ([1206](#)), `Pi` ([1286](#)), `Exp` ([1220](#)), `Ln` ([1251](#))

Listing: `./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.
```

36.11.376 SizeOf

Synopsis: Return size 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` ([1185](#))

Listing: `./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.
```

36.11.377 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` ([1252](#))

Errors: In case of an error, 0 is returned.

See also: `FindResource` ([1225](#)), `FreeResource` ([1229](#)), `LoadResource` ([1252](#)), `LockResource` ([1253](#)), `UnlockResource` ([1333](#)), `FreeResource` ([1229](#))

36.11.378 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` ([1250](#)), `SetLength` ([1309](#))

Listing: `./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.

36.11.379 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` ([1320](#))

36.11.380 Sptr

Synopsis: Return current stack pointer

Declaration: `function Sptr : Pointer`

Visibility: default

Description: `Sptr` returns the current stack pointer.

Errors: None.

See also: `SSeg` ([1318](#))

Listing: `./refex/ex64.pp`

```

program Example64;

  { Program to demonstrate the sptr function. }

  var p: ptnuint;

  begin
    p:=ofs(sptr); { P Contains now the current stack position. }
  end.

```

36.11.381 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` ([1318](#)), `Ln` ([1251](#)), `Exp` ([1220](#))

Listing: `./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.

```

36.11.382 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` ([1317](#)), `Ln` ([1251](#)), `Exp` ([1220](#))

Listing: `./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.
```

36.11.383 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` ([1317](#))

Listing: `./refex/ex67.pp`

Program `Example67;`

{ Program to demonstrate the SSeg function. }

Var `W : Longint;`

```
begin
  W:=SSeg;
end.
```

36.11.384 StackTop

Synopsis: Top location of the stack.

Declaration: `function StackTop : Pointer`

Visibility: default

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` ([1181](#)) is returned.

See also: `DefaultSystemCodePage` ([1181](#)), `StringElementSize` ([1320](#)), `StringRefCount` ([1321](#)), `SetCodePage` ([1308](#))

36.11.387 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` ([1319](#)), `StringRefCount` ([1321](#))

36.11.388 StringOfChar

Synopsis: Return a string consisting of 1 character repeated N times.

Declaration: `function StringOfChar(c: AnsiChar; l: SizeInt) : AnsiString`

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, l);
FillChar(Pointer(StringOfChar)^, Length(StringOfChar), c);
```

Errors: None.

See also: `SetLength` ([1309](#))

Listing: `./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.

36.11.389 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` ([1319](#)), `StringElementSize` ([1320](#))

36.11.390 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` ([1187](#))

36.11.391 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` ([1332](#)), `UnicodeCharLenToString` ([1331](#))

36.11.392 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 an ansistring `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` ([1339](#)), `WideCharToStrVar` ([1339](#)), `WideCharLenToStrVar` ([1338](#)), `WideCharLenToString` ([1338](#))

36.11.393 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.

36.11.394 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.

36.11.395 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` ([1284](#)).

Errors: Run-time error 201 is generated when the result is out of range.

See also: `Ord` ([1284](#)), `Pred` ([1288](#)), `High` ([1237](#)), `Low` ([1253](#))

36.11.396 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` ([1296](#)), `KillThread` ([1249](#))

36.11.397 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` ([1252](#)), `Hi` ([1236](#))

Listing: `./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.
```

36.11.398 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` ([1236](#)), `lo` ([1252](#)), `swap` ([1323](#)), `BEToN` ([1191](#)), `NToBE` ([1257](#)), `NToLE` ([1257](#)), `LEToN` ([1251](#))

36.11.399 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` ([1186](#))

See also: `AllocMem` ([1186](#))

36.11.400 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` ([1124](#))

36.11.401 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.

36.11.402 SysFlushStdIO

Synopsis:

Declaration: `procedure SysFlushStdIO`

Visibility: `default`

Description:

36.11.403 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` ([1228](#))

See also: `FreeMem` ([1228](#))

36.11.404 SysFreememSize

Synopsis: System memory manager free routine.

Declaration: `function SysFreememSize(p: pointer; Size: PtrUInt) : PtrUInt`

Visibility: `default`

Description: `SysFreeSize` is the system memory manager implementation for `FreeMem` ([1228](#))

See also: `MemSize` ([1254](#))

36.11.405 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` ([1230](#)) function.

Errors: None. The result of this function is bogus information if the current heapmanager is not the standard FPC heapmanager.

See also: `GetFPCHeapStatus` ([1230](#))

36.11.406 SysGetHeapStatus

Synopsis: System implementation of `GetHeapStatus` ([1231](#))

Declaration: `function SysGetHeapStatus : THeapStatus`

Visibility: `default`

Description: `SysGetHeapStatus` is the system implementation of the `GetHeapStatus` ([1231](#)) call.

See also: `GetHeapStatus` ([1231](#))

36.11.407 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` ([1231](#))

See also: `GetMem` ([1231](#)), `GetMemory` ([1231](#))

36.11.408 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.

36.11.409 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` ([1191](#))

36.11.410 SysInitStdIO

Synopsis: Initialize standard input and output.

Declaration: `procedure SysInitStdIO`

Visibility: default

Description: `SysInitStdIO` initializes the standard input and output files: `Output` ([1182](#)), `Input` ([1182](#)) and `StdErr` ([1183](#)). This routine is called by the initialization code of the system unit, there should be no need to call it directly.

36.11.411 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` ([1254](#))

See also: `MemSize` ([1254](#))

36.11.412 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` ([1294](#)).

See also: `ReAllocMem` ([1294](#))

36.11.413 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.

36.11.414 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` ([1161](#))

36.11.415 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` ([1294](#)), `SysReAllocMem` ([1327](#))

See also: `SysReAllocMem` ([1327](#)), `ReAllocMem` ([1294](#))

36.11.416 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` ([1328](#))

36.11.417 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` ([1328](#))

36.11.418 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` ([1323](#)), `ResumeThread` ([1296](#)), `KillThread` ([1249](#))

36.11.419 ToSingleByteFileSystemEncodedFileName

Synopsis: Convert string to encoding for use in single-byte filesystem 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 filesystem API.

This routine is simply an auxiliary routine, which converts the argument to a single-byte string using `DefaultFileSystemCodePage` (1180) as a codepage.

See also: `DefaultFileSystemCodePage` (1180)

36.11.420 `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: None.

See also: `Frac` (1228), `Int` (1245), `Round` (1300)

Listing: `./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.
```

36.11.421 `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 `{ $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: `Append` (1186), `Filepos` (1220), `Seek` (1304)

Listing: `./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.

```

36.11.422 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.

36.11.423 TypeInfo

Synopsis: Return pointer to type information for type

Declaration: `function TypeInfo(const T: AnyType) : Pointer`

Visibility: default

Description: `Default` 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` ([1118](#))

36.11.424 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` ([1332](#))

36.11.425 UCS4StringToWideString

Synopsis:

Declaration: `function UCS4StringToWideString(const s: UCS4String) : WideString`

Visibility: default

Description:

36.11.426 Unassigned

Synopsis: Unassigned variant.

Declaration: `function Unassigned : Variant`

Visibility: default

36.11.427 UnicodeCharLenToString

Synopsis: Convert a memory buffer with unicode characters to an ansistring

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 an ansistring 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` ([1321](#)), `UnicodeCharToString` ([1332](#))

36.11.428 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` ([1331](#)).

Errors: No checking is done to see if the pointer `S` or length `len` are valid.

See also: `StringToUnicodeChar` ([1321](#)), `UnicodeCharToString` ([1332](#)), `UnicodeCharLenToString` ([1331](#)), `UnicodeCharToStrVar` ([1332](#))

36.11.429 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 an `AnsiString` value. It simply calls `UnicodeCharLenToString` (1331) 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` (1321), `UnicodeCharLenToString` (1331), `WidestringManager` (1183)

36.11.430 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` (1332).

Errors: No checking is done to see if the pointer `S` is valid.

See also: `StringToUnicodeChar` (1321), `UnicodeCharToString` (1332), `UnicodeCharLenToString` (1331), `UnicodeCharToString` (1332)

36.11.431 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` (1330)

36.11.432 UnicodeToUtf8

Synopsis:

```
Declaration: function UnicodeToUtf8(Dest: PChar;Source: PUnicodeChar;
                                   MaxBytes: SizeInt) : SizeInt
              function UnicodeToUtf8(Dest: PChar;MaxDestBytes: SizeUInt;
                                   Source: PUnicodeChar;SourceChars: SizeUInt)
                                   : SizeUInt
```

Visibility: default

Description:

36.11.433 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.

36.11.434 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` ([1225](#)), `FreeResource` ([1229](#)), `SizeofResource` ([1316](#)), `LoadResource` ([1252](#)), `lockResource` ([1253](#)), `FreeResource` ([1229](#))

36.11.435 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` ([1285](#))

36.11.436 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: `Uppcase` 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.

Errors: None.

See also: `Lowercase` ([1254](#))

Listing: `./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.
```

36.11.437 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.

36.11.438 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` (1336) function to do the actual work.

For this function to work, a widestring manager must be installed.

See also: `UTF8Encode` (1335), `Utf8ToAnsi` (1335), `SetWideStringManager` (1314), `Utf8ToUnicode` (1336)

36.11.439 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` (1332) function to do the actual work.

For this function to work, a widestring manager must be installed.

See also: `UTF8Decode` (1335), `Utf8ToAnsi` (1335), `UnicodeToUtf8` (1332), `SetWideStringManager` (1314)

36.11.440 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` (1335), `UTF8Decode` (1335), `SetWideStringManager` (1314)

36.11.441 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`.

The function returns the number of copied widestring characters.

For this function to work, a widestring manager must be installed.

Errors: On error, -1 is returned.

See also: `UTF8Encode` ([1335](#)), `UTF8Decode` ([1335](#)), `Utf8ToAnsi` ([1335](#)), `SetWideStringManager` ([1314](#))

36.11.442 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 type `Longint`, `Real` and `Byte` or any enumerated type. If the conversion isn't 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.

For hexadecimal values, the prefix '0x' or 'x' (case insensitive) may be used as well.

The conversion to enumerated exists only as of version 2.3.1 (or later) of the compiler.

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` ([1319](#))

Listing: `./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.

```

36.11.443 VarArrayGet

Synopsis:

Declaration: `function VarArrayGet(const A: Variant;const Indices: Array of LongInt)
: Variant`

Visibility: default

Description:

36.11.444 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` ([1337](#))

36.11.445 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.

36.11.446 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 pre-defined `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.

36.11.447 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` ([1213](#)), `KillThread` ([1249](#))

36.11.448 WideCharLenToString

Synopsis: Convert a length-limited array of widechar to an ansistring

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 ansistring, and returns the ansistring.

Errors: No validity checking is performed on `S`. Passing an invalid pointer may lead to access violations.

See also: `StringToWideChar` ([1322](#)), `WideCharToString` ([1339](#)), `WideCharToStrVar` ([1339](#)), `WideCharLenToStrVar` ([1338](#))

36.11.449 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, and returns the ansistring in `Dest`.

Errors: No validity checking is performed on `Src`. Passing an invalid pointer may lead to access violations.

See also: `StringToWideChar` (1322), `WideCharToString` (1339), `WideCharToStrVar` (1339), `WideCharLenToString` (1338)

36.11.450 WideCharToString

Synopsis: Convert a null-terminated array of widechar to an ansistring

Declaration: `function WideCharToString(S: PWideChar) : UnicodeString`

Visibility: default

Description: `WideCharToString` converts the null-terminated widechar array `S` to an ansistring, and returns the ansistring.

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` (1322), `WideCharToStrVar` (1339), `WideCharLenToStrVar` (1338), `WideCharLenToString` (1338)

36.11.451 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, and returns the ansistring 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` (1322), `WideCharToString` (1339), `WideCharToStrVar` (1339), `WideCharLenToString` (1338)

36.11.452 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.

36.11.453 Write

Synopsis: Write variable to a text file

Declaration: `procedure Write(Args: Arguments)`
`procedure Write(var F: Text; Args: Arguments)`

Visibility: default

Description: `Write` writes the contents of the variables `V1`, `V2` 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.

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.

Errors: If an error occurs, a run-time error is generated. This behavior can be controlled with the `{SI}` switch.

See also: `WriteLn` ([1340](#)), `Read` ([1291](#)), `ReadLn` ([1292](#)), `Blockwrite` ([1193](#))

36.11.454 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` ([1291](#)), `ReadDependencyBarrier` ([1292](#)), `ReadWriteBarrier` ([1293](#))

36.11.455 WriteLn

Synopsis: Write variable to a text file and append newline

Declaration: `procedure Writeln(Args: Arguments)`
`procedure WriteLn(var F: Text; Args: Arguments)`

Visibility: default

Description: `WriteLn` does the same as `Write` (1340) 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` (1140) 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.

Errors: If an error occurs, a run-time error is generated. This behavior can be controlled with the `{SI}` switch.

See also: `Write` (1340), `Read` (1291), `ReadLn` (1292), `Blockwrite` (1193), `slinebreak` (1140)

Listing: `./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.
```

36.11.456 WriteStr

Synopsis: Write variables to a string

Declaration: `procedure WriteStr(out S: string;Args: Arguments)`

Visibility: default

Description: `WriteStr` behaves like `Write` (1340), 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` ([1340](#)).

See also: `Write` ([1340](#)), `ReadStr` ([1292](#)), `Read` ([1291](#))

36.12 IDispatch

36.12.1 Description

`IDispatch` is the pascal definition of the Windows Dispatch interface definition.

See also: `IUnknown` ([1345](#))

36.12.2 Method overview

Page	Property	Description
1342	<code>GetIDsOfNames</code>	Return IDs of named procedures
1342	<code>GetTypeInfo</code>	Return type information about properties
1342	<code>GetTypeInfoCount</code>	Return number of properties.
1343	<code>Invoke</code>	Invoke a dispatch method

36.12.3 IDispatch.GetTypeInfoCount

Synopsis: Return number of properties.

Declaration: `function GetTypeInfoCount(out count: LongInt) : HRESULT`

Visibility: default

36.12.4 IDispatch.GetTypeInfo

Synopsis: Return type information about properties

Declaration: `function GetTypeInfo(Index: LongInt; LocaleID: LongInt; out TypeInfo) : HRESULT`

Visibility: default

36.12.5 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.

36.12.6 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

36.13 IEnumerable

36.13.1 Description

`IEnumerable` can be used to get an enumerator from a class. If a class implements `IEnumerable`, it can return an enumerator interface `IEnumerator` ([1343](#)).

See also: `IEnumerator` ([1343](#))

36.13.2 Method overview

Page	Property	Description
1343	<code>GetEnumerator</code>	Return an enumerator interface for this class

36.13.3 IEnumerable.GetEnumerator

Synopsis: Return an enumerator interface for this class

Declaration: `function GetEnumerator : IEnumerator`

Visibility: default

Description: `GetEnumerator` returns a new `IEnumerator` ([1343](#)) 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` ([1343](#))

36.14 IEnumerator

36.14.1 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` ([1343](#))

36.14.2 Method overview

Page	Property	Description
1344	<code>GetCurrent</code>	Returns the current element in the iteration cycle
1344	<code>MoveNext</code>	Move to the next value
1345	<code>Reset</code>	Reset the pointer

36.14.3 Property overview

Page	Property	Access	Description
1345	<code>Current</code>	<code>r</code>	Return the current item

36.14.4 `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` ([1344](#)) 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` ([1344](#)), `IEnumerator.Reset` ([1345](#))

36.14.5 `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` (1344) will be called to retrieve the item.

See also: `IEnumerator.Reset` (1345), `IEnumerator.GetCurrent` (1344)

36.14.6 `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` (1344), `IEnumerator.MoveNext` (1344)

36.14.7 `IEnumerator.Current`

Synopsis: Return the current item

Declaration: `Property Current : TObject`

Visibility: default

Access: Read

Description: `Current` simply is the redefinition of `IEnumerator.GetCurrent` (1344) as a property. It is read-only.

See also: `IEnumerator.GetCurrent` (1344), `IEnumerator.MoveNext` (1344)

36.15 `IInvokable`

36.15.1 `Description`

`IInvokable` is a descendent of `IInterface` (1150), compiled in the `{ $M+ }` state, so Run-Time Type Information (RTTI) is generated for it.

See also: `IDispatch` (1342), `IInterface` (1150)

36.16 `IUnknown`

36.16.1 `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` (1150), `IDispatch` (1342), `IInvokable` (1345)

36.16.2 Method overview

Page	Property	Description
1346	QueryInterface	Return pointer to VMT table of interface
1346	_AddRef	Increase reference count of the interface
1346	_Release	Decrease reference count of the interface

36.16.3 IUnknown.QueryInterface

Synopsis: Return pointer to VMT table of interface

Declaration: `function QueryInterface(const iid: TGuid;out obj) : LongInt`

Visibility: default

36.16.4 IUnknown._AddRef

Synopsis: Increase reference count of the interface

Declaration: `function _AddRef : LongInt`

Visibility: default

See also: IUnknown._Release ([1346](#))

36.16.5 IUnknown._Release

Synopsis: Decrease reference count of the interface

Declaration: `function _Release : LongInt`

Visibility: default

See also: IUnknown._AddRef ([1346](#))

36.17 TAggregatedObject

36.17.1 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 ([1347](#)) property. In effect, the reference count of the aggregated object is the same as that of it's 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 ([1347](#)), Controller ([1347](#))

36.17.2 Method overview

Page	Property	Description
1347	Create	Create a new instance of TAggregatedObject

36.17.3 Property overview

Page	Property	Access	Description
1347	Controller	r	Controlling instance

36.17.4 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` ([1347](#)) 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` ([1347](#))

36.17.5 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` ([1347](#))

36.18 TContainedObject

36.18.1 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` ([1150](#))

36.18.2 Interfaces overview

Page	Property	Description
1150	IInterface	Basic interface for all COM based interfaces

36.19 TInterfacedObject

36.19.1 Description

TInterfacedObject is a descendent of TObject (1349) which implements the IUnknown (1345) interface. It can be used as a base class for all classes which need reference counting.

See also: IUnknown (1345), TObject (1349)

36.19.2 Interfaces overview

Page	Property	Description
1345	IUnknown	Basic interface for all COM-based interfaces

36.19.3 Method overview

Page	Property	Description
1348	AfterConstruction	Handle reference count properly.
1348	BeforeDestruction	Check reference count.
1349	NewInstance	Create a new instance

36.19.4 Property overview

Page	Property	Access	Description
1349	RefCount	r	Return the current reference count

36.19.5 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` (1348)

36.19.6 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` (1348)

36.19.7 TInterfacedObject.NewInstance

Synopsis: Create a new instance

Declaration: `class function NewInstance; Override`

Visibility: `public`

Description: `NewInstance` initializes a new instance of `TInterfacedObject` ([1348](#))

Errors: None.

36.19.8 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` ([1345](#)). When it reaches zero, the class instance is destroyed.

See also: `IUnknown` ([1345](#))

36.20 TObject

36.20.1 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` ([1161](#))

36.20.2 Method overview

Page	Property	Description
1355	AfterConstruction	Method called after the constructor was called.
1356	BeforeDestruction	Method called before the destructor is called.
1353	ClassInfo	Return a pointer to the type information for this class.
1353	ClassName	Return the current class name.
1353	ClassNameIs	Check whether the class name equals the given name.
1354	ClassParent	Return the parent class.
1353	ClassType	Return a "class of" pointer for the current class
1352	CleanupInstance	Finalize the class instance.
1350	Create	TObject Constructor
1352	DefaultHandler	Default handler for integer message handlers.
1356	DefaultHandlerStr	Default handler for string messages.
1351	Destroy	TObject destructor.
1356	Dispatch	Dispatch an integer message
1356	DispatchStr	Dispatch a string message.
1359	Equals	Check if two objects are equal.
1355	FieldAddress	Return the address of a field.
1352	Free	Check for Nil and call destructor.
1351	FreeInstance	Clean up instance and free the memory reserved for the instance.
1359	GetHashCode	Return a hash code for the object
1357	GetInterface	Return a reference to an interface
1357	GetInterfaceByStr	Return an interface based on its GUID
1358	GetInterfaceEntry	Return the interface table entry by GUID
1358	GetInterfaceEntryByStr	Return the interface table entry by string
1358	GetInterfaceTable	Return a pointer to the table of implemented interfaces for a class
1357	GetInterfaceWeak	Get a reference to an interface, not increasing the reference count
1354	InheritsFrom	Check whether class is an ancestor.
1352	InitInstance	Initialize a new class instance.
1354	InstanceSize	Return the size of an instance.
1355	MethodAddress	Return the address of a method
1355	MethodName	Return the name of a method.
1351	newinstance	Allocate memory on the heap for a new instance
1351	SafeCallException	Handle exception object
1354	StringMessageTable	Return a pointer to the string message table.
1359	ToString	Return a string representation for the object
1358	UnitName	Unit name

36.20.3 TObject.Create

Synopsis: TObject Constructor

Declaration: constructor Create

Visibility: public

Description: Create creates a new instance of TObject. Currently it does nothing. It is also not virtual, so there is in principle no need to call it directly.

See also: Destroy ([1351](#))

36.20.4 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 (1352) method, because that one will check first if Self is different from Nil.

To clean up an instance and reset the refence to the instance, it is best to use the FreeAndNil (1459) function.

See also: Create (1350), Free (1352)

36.20.5 TObject.newinstance

Synopsis: Allocate memory on the heap for a new instance

Declaration: class function newInstance; 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 (1352). The function returns the newly initialized instance.

Errors: If not enough memory is available, a Nil pointer may be returned, or an exception may be raised.

See also: Create (1350), InitInstance (1352), InstanceSize (1354), FreeInstance (1351)

36.20.6 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 (1351), InitInstance (1352), NewInstance (1351)

36.20.7 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.

36.20.8 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` (1356) method.

See also: `Dispatch` (1356), `DefaultHandlerStr` (1356)

36.20.9 TObject.Free

Synopsis: Check for `Nil` and call destructor.

Declaration: `procedure Free`

Visibility: `public`

Description: `Free` will check the `Self` pointer and calls `Destroy` (1351) 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` (1459) should be called.

Errors: None.

See also: `Destroy` (1351), `FreeAndNil` (1459)

36.20.10 TObject.InitInstance

Synopsis: Initialize a new class instance.

Declaration: `class function InitInstance(instance: pointer)`

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` (1351), `Create` (1350)

36.20.11 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 its constructor.

Errors: None.

See also: `Destroy` (1351), `Free` (1352), `InitInstance` (1352)

36.20.12 TObject.ClassType

Synopsis: Return a "class of" pointer for the current class

Declaration: `class function ClassType`

Visibility: public

Description: `ClassType` returns a `TClass` (1161) class type reference for the current class.

See also: `TClass` (1161), `ClassInfo` (1353), `ClassName` (1353)

36.20.13 TObject.ClassInfo

Synopsis: Return a pointer to the type information for this class.

Declaration: `class function ClassInfo`

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.

36.20.14 TObject.ClassName

Synopsis: Return the current class name.

Declaration: `class function ClassName`

Visibility: public

Description: `ClassName` returns the class name for the current class, in all-uppercase letters. To check for the class name, use the `ClassNameIs` (1353) class method.

Errors: None.

See also: `ClassInfo` (1353), `ClassType` (1353), `ClassNameIs` (1353)

36.20.15 TObject.ClassNameIs

Synopsis: Check whether the class name equals the given name.

Declaration: `class function ClassNameIs(const name: string)`

Visibility: public

Description: `ClassNameIs` checks whether `Name` equals the class name. It takes of case sensitivity, i.e. it converts both names to uppercase before comparing.

See also: `ClassInfo` (1353), `ClassType` (1353), `ClassName` (1353)

36.20.16 TObject.ClassParent

Synopsis: Return the parent class.

Declaration: `class function ClassParent`

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` ([1353](#)), `ClassType` ([1353](#)), `ClassNameIs` ([1353](#))

36.20.17 TObject.InstanceSize

Synopsis: Return the size of an instance.

Declaration: `class function InstanceSize`

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` ([1352](#)), `ClassName` ([1353](#)), `ClassInfo` ([1353](#)), `ClassType` ([1353](#))

36.20.18 TObject.InheritsFrom

Synopsis: Check whether class is an ancestor.

Declaration: `class function InheritsFrom(aClass: TClass)`

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` ([1353](#)), `ClassInfo` ([1353](#)), `ClassType` ([1353](#)), `TClass` ([1161](#))

36.20.19 TObject.StringMessageTable

Synopsis: Return a pointer to the string message table.

Declaration: `class function StringMessageTable`

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` ([1356](#)) method.

Errors: If there are no string message handlers, `nil` is returned.

See also: `DispatchStr` ([1356](#)), `Dispatch` ([1356](#))

36.20.20 TObject.MethodAddress

Synopsis: Return the address of a method

Declaration: `class function MethodAddress(const name: shortstring)`

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` ([1355](#)), `FieldAddress` ([1355](#))

36.20.21 TObject.MethodName

Synopsis: Return the name of a method.

Declaration: `class function MethodName(address: CodePointer)`

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` ([1355](#)), `FieldAddress` ([1355](#))

36.20.22 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` ([1355](#)), `MethodName` ([1355](#))

36.20.23 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` ([1356](#)), `Create` ([1350](#))

36.20.24 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` ([1355](#)), `Destroy` ([1351](#)), `Free` ([1352](#))

36.20.25 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` ([1356](#)), `Dispatch` ([1356](#)), `DefaultHandler` ([1352](#))

36.20.26 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` ([1352](#)) method, which can be overridden by descendent classes to add custom handling of messages.

See also: `DispatchStr` ([1356](#)), `DefaultHandler` ([1352](#))

36.20.27 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` ([1356](#)) is called.

Errors: None.

See also: [DefaultHandlerStr \(1356\)](#), [Dispatch \(1356\)](#), [DefaultHandler \(1352\)](#)

36.20.28 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 \(1357\)](#)

36.20.29 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 \(1358\)](#), [TObject.GetInterfaceEntryByStr \(1358\)](#)

36.20.30 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 \(1357\)](#), but unlike the latter, it will not increase the reference count of the interface.

See also: [TObject.GetInterface \(1357\)](#)

36.20.31 TObject.GetInterfaceEntry

Synopsis: Return the interface table entry by GUID

Declaration: `class function GetInterfaceEntry(const iid: TGuid)`

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` ([1357](#)), `TObject.GetInterfaceEntryByStr` ([1358](#))

36.20.32 TObject.GetInterfaceEntryByStr

Synopsis: Return the interface table entry by string

Declaration: `class function GetInterfaceEntryByStr(const iidstr: shortstring)`

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` ([1357](#)), `TObject.GetInterfaceEntry` ([1358](#))

36.20.33 TObject.GetInterfaceTable

Synopsis: Return a pointer to the table of implemented interfaces for a class

Declaration: `class function GetInterfaceTable`

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` ([1357](#)), `TObject.GetInterfaceEntry` ([1358](#))

36.20.34 TObject.UnitName

Synopsis: Unit name

Declaration: `class function UnitName`

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.

36.20.35 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` ([1359](#)), `TObject.ToString` ([1359](#))

36.20.36 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` ([1359](#)), `TObject.Equals` ([1359](#))

36.20.37 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` ([1359](#)), `TObject.Equals` ([1359](#))

Chapter 37

Reference for unit 'sysutils'

37.1 Used units

Table 37.1: Used units by unit 'sysutils'

Name	Page
errors	??
Linux	746
sysconst	??
System	1118
Unix	1589
unixtype	1623

37.2 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.

37.3 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 unices, 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.

37.4 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` (1180) 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 37.2:

Name	Description
<code>FileCreate</code> (1432)	Create a new file and return a handle to it.
<code>FileOpen</code> (1436)	Open an existing file and return a filehandle
<code>FileExists</code> (1433)	Check whether a particular file exists in the filesystem.
<code>DirectoryExists</code> (1418)	Check whether a directory exists in the file system.
<code>FileSetDate</code> (1439)	Set the date of a file.
<code>FileGetAttr</code> (1434)	Return attributes of a file.
<code>FileSetAttr</code> (1439)	Set the attributes of a file.
<code>DeleteFile</code> (1418)	Delete a file from the filesystem.
<code>RenameFile</code> (1477)	Rename a file.
<code>FileSearch</code> (1437)	Search for a file in a path.
<code>ExeSearch</code> (1425)	Search for an executable
<code>FindFirst</code> (1441)	Start a file search and return a findhandle
<code>FindNext</code> (1442)	Find the next entry in a findhandle.
<code>FindClose</code> (1440)	Close a find handle
<code>FileIsReadOnly</code> (1436)	Check whether a file is read-only.
<code>GetCurrentDir</code> (1460)	Return the current working directory of the application.
<code>SetCurrentDir</code> (1480)	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 37.3:

Name	Description
ChangeFileExt (1405)	Change the extension of a filename.
ExtractFilePath (1429)	Extract the path from a filename.
ExtractFileDrive (1428)	Extract the drive part from a filename.
ExtractFileName (1429)	Extract the filename part from a full path filename.
ExtractFileExt (1429)	Return the extension from a filename.
ExtractFileDir (1428)	Extract the drive and directory part of a filename.
ExtractShortPathName (1430)	Returns a 8.3 path name
ExpandFileName (1426)	Expand a relative filename to an absolute filename.
ExpandFileNameCase (1426)	Expand a filename entered as case insensitive to the full path as stored on the disk.
ExtractRelativepath (1430)	Extract a relative path from a filename, given a base directory.
ExpandUNCFileName (1427)	Expand a relative filename to an absolute UNC filename.
IncludeTrailingPathDelimiter (1468)	Add trailing directory separator to a pathname, if needed.
IncludeTrailingBackslash (1468)	Add trailing directory separator to a pathname, if needed.
ExcludeTrailingBackslash (1424)	Strip trailing directory separator from a pathname, if needed.
ExcludeTrailingPathDelimiter (1424)	Strip trailing directory separator from a pathname, if needed.
IncludeLeadingPathDelimiter (1467)	Prepend a path delimiter if there is not already one.
ExcludeLeadingPathDelimiter (1424)	Strip the leading path delimiter of a path
IsPathDelimiter (1472)	Is the character at the given position a pathdelimiter ?
DoDirSeparators (1420)	Convert known directory separators to the current directory separator.
SetDirSeparators (1480)	Set the directory separators to the known directory separators.
GetDirs (1461)	Return a list of directory names from a path.
ConcatPaths (1409)	Concatenate an array of paths to form a single path
GetEnvironmentVariable (1462)	Return the value of an environment variable.

37.5 Miscellaneous conversion routines

Functions for various conversions.

Table 37.4:

Name	Description
BCDToInt (1403)	Convert BCD number to integer
CompareMem (1406)	Compare two memory regions
FloatToStrF (1445)	Convert float to formatted string
FloatToStr (1444)	Convert float to string
FloatToText (1447)	Convert float to string
FormatFloat (1457)	Format a floating point value
GetDirs (1461)	Split string in list of directories
IntToHex (1469)	return hexadecimal representation of integer
IntToStr (1470)	return decumal representation of integer
StrToIntDef (1502)	Convert string to integer with default value
StrToInt (1500)	Convert string to integer
StrToFloat (1499)	Convert string to float
TextToFloat (1506)	Convert null-terminated string to float

37.6 Date/time routines

Functions for date and time handling.

Table 37.5:

Name	Description
<code>DateTimeToFileDate</code> (1412)	Convert <code>DateTime</code> type to file date
<code>DateTimeToStr</code> (1413)	Construct string representation of <code>DateTime</code>
<code>DateTimeToString</code> (1413)	Construct string representation of <code>DateTime</code>
<code>DateTimeToSystemTime</code> (1414)	Convert <code>DateTime</code> to system time
<code>DateTimeToTimeStamp</code> (1415)	Convert <code>DateTime</code> to timestamp
<code>DateToStr</code> (1415)	Construct string representation of date
<code>Date</code> (1412)	Get current date
<code>DayOfWeek</code> (1416)	Get day of week
<code>DecodeDate</code> (1416)	Decode <code>DateTime</code> to year month and day
<code>DecodeTime</code> (1417)	Decode <code>DateTime</code> to hours, minutes and seconds
<code>EncodeDate</code> (1421)	Encode year, day and month to <code>DateTime</code>
<code>EncodeTime</code> (1422)	Encode hours, minutes and seconds to <code>DateTime</code>
<code>FormatDateTime</code> (1457)	Return string representation of <code>DateTime</code>
<code>IncMonth</code> (1468)	Add 1 to month
<code>IsLeapYear</code> (1471)	Determine if year is leap year
<code>MSecsToTimeStamp</code> (1474)	Convert nr of milliseconds to timestamp
<code>Now</code> (1475)	Get current date and time
<code>StrToDateTime</code> (1498)	Convert string to <code>DateTime</code>
<code>StrToDate</code> (1497)	Convert string to date
<code>StrToTime</code> (1503)	Convert string to time
<code>SystemTimeToDateTime</code> (1505)	Convert system time to datetime
<code>TimeStampToDateTime</code> (1508)	Convert time stamp to <code>DateTime</code>
<code>TimeStampToMSecs</code> (1508)	Convert <code>TimeStamp</code> to number of milliseconds
<code>TimeToStr</code> (1509)	return string representation of <code>Time</code>
<code>Time</code> (1507)	Get current time

37.7 FileName handling routines

Functions for file manipulation.

Table 37.6:

Name	Description
AnsiCompareFileName (1388)	Compare 2 filenames
AnsiLowerCaseFileName (1393)	Create lowercase filename
AnsiUpperCaseFileName (1401)	Create uppercase filename
AddDisk (1387)	Add disk to list of disk drives
ChangeFileExt (1405)	Change extension of file name
CreateDir (1410)	Create a directory
DeleteFile (1418)	Delete a file
DiskFree (1419)	Free space on disk
DiskSize (1419)	Total size of disk
ExpandFileName (1426)	Create full file name
ExpandFileNameCase (1426)	Create full file name case insensitively
ExpandUNCFileName (1427)	Create full UNC file name
ExtractFileDir (1428)	Extract drive and directory part of filename
ExtractFileDrive (1428)	Extract drive part of filename
ExtractFileExt (1429)	Extract extension part of filename
ExtractFileName (1429)	Extract name part of filename
ExtractFilePath (1429)	Extract path part of filename
ExtractRelativePath (1430)	Construct relative path between two files
FileAge (1431)	Return file age
FileDateToDateTime (1433)	Convert file date to system date
FileExists (1433)	Determine whether a file exists on disk
FileGetAttr (1434)	Get attributes of file
FileGetDate (1435)	Get date of last file modification
FileSearch (1437)	Search for file in path
FileSetAttr (1439)	Get file attributes
FileSetDate (1439)	Get file dates
FindFirst (1441)	Start finding a file
FindNext (1442)	Find next file
GetCurrentDir (1460)	Return current working directory
RemoveDir (1477)	Remove a directory from disk
RenameFile (1477)	Rename a file on disk
SameFileName (1479)	Check whether 2 filenames are the same
SetCurrentDir (1480)	Set current working directory
SetDirSeparators (1480)	Set directory separator characters
FindClose (1440)	Stop searching a file
DoDirSeparators (1420)	Replace directory separator characters

37.8 File input/output routines

Functions for reading/writing to file.

Table 37.7:

Name	Description
FileCreate (1432)	Create a file and return handle
FileOpen (1436)	Open file and return handle
FileRead (1437)	Read from file
FileSeek (1438)	Set file position
FileTruncate (1440)	Truncate file length
FileWrite (1440)	Write to file
FileClose (1431)	Close file handle

37.9 PChar related functions

Most PChar functions are the same as their counterparts in the **STRINGS** unit. The following functions are the same :

1. StrCat (1483) : Concatenates two PChar strings.
2. StrComp (1484) : Compares two PChar strings.
3. StrCopy (1484) : Copies a PChar string.
4. StrECopy (1485) : Copies a PChar string and returns a pointer to the terminating null byte.
5. StrEnd (1485) : Returns a pointer to the terminating null byte.
6. StrIComp (1487) : Case insensitive compare of 2 PChar strings.
7. StrLCat (1488) : Appends at most L characters from one PChar to another PChar.
8. StrLComp (1489) : Case sensitive compare of at most L characters of 2 PChar strings.
9. StrLCopy (1489) : Copies at most L characters from one PChar to another.
10. StrLen (1490) : Returns the length (exclusive terminating null byte) of a PChar string.
11. StrLIComp (1491) : Case insensitive compare of at most L characters of 2 PChar strings.
12. StrLower (1491) : Converts a PChar to all lowercase letters.
13. StrMove (1492) : Moves one PChar to another.
14. StrNew (1493) : Makes a copy of a PChar on the heap, and returns a pointer to this copy.
15. StrPos (1494) : Returns the position of one PChar string in another?
16. StrRScan (1495) : returns a pointer to the last occurrence of on PChar string in another one.
17. StrScan (1495) : returns a pointer to the first occurrence of on PChar string in another one.
18. StrUpper (1504) : Converts a PChar to all uppercase letters.

The subsequent functions are different from their counterparts in **STRINGS**, although the same examples can be used.

37.10 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

mm month (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)

The date and time separators are taken from the DefaultFormatSettings (1383) record, unless a TFormatSettings (1379) record is passed to the FormatDateTime (1457) 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 (1383), TFormatSettings (1379), FormatDateTime (1457)

37.11 Formatting strings

Functions for formatting strings.

Table 37.8:

Name	Description
AdjustLineBreaks (1388)	Convert line breaks to line breaks for system
FormatBuf (1456)	Format a buffer
Format (1449)	Format arguments in string
FmtStr (1448)	Format buffer
QuotedStr (1476)	Quote a string
StrFmt (1486)	Format arguments in a string
StrLFmt (1490)	Format maximum L characters in a string
TrimLeft (1510)	Remove whitespace at the left of a string
TrimRight (1510)	Remove whitespace at the right of a string
Trim (1509)	Remove whitespace at both ends of a string

37.12 String functions

Functions for handling strings.

Table 37.9:

Name	Description
AnsiCompareStr (1389)	Compare two strings
AnsiCompareText (1390)	Compare two strings, case insensitive
AnsiExtractQuotedStr (1391)	Removes quotes from string
AnsiLastChar (1392)	Get last character of string
AnsiLowerCase (1392)	Convert string to all-lowercase
AnsiQuotedStr (1393)	Quotes a string
AnsiStrComp (1394)	Compare strings case-sensitive
AnsiStrIComp (1395)	Compare strings case-insensitive
AnsiStrLComp (1397)	Compare L characters of strings case sensitive
AnsiStrLIComp (1397)	Compare L characters of strings case insensitive
AnsiStrLastChar (1396)	Get last character of string
AnsiStrLower (1398)	Convert string to all-lowercase
AnsiStrUpper (1400)	Convert string to all-uppercase
AnsiUpperCase (1400)	Convert string to all-uppercase
AppendStr (1401)	Append 2 strings
AssignStr (1402)	Assign value of strings on heap
CompareStr (1407)	Compare two strings case sensitive
CompareText (1408)	Compare two strings case insensitive
DisposeStr (1420)	Remove string from heap
IsValidIdent (1472)	Is string a valid pascal identifier
LastDelimiter (1473)	Last occurrence of character in a string
LeftStr (1473)	Get first N characters of a string
LoadStr (1474)	Load string from resources
LowerCase (1474)	Convert string to all-lowercase
NewStr (1475)	Allocate new string on heap
RightStr (1478)	Get last N characters of a string
StrAlloc (1482)	Allocate memory for string
StrBufSize (1482)	Reserve memory for a string
StrDispose (1485)	Remove string from heap
StrPas (1493)	Convert PChar to pascal string
StrPCopy (1494)	Copy pascal string
StrPLCopy (1494)	Copy N bytes of pascal string
UpperCase (1516)	Convert string to all-uppercase

37.13 Constants, types and variables

37.13.1 Constants

`ConfigExtension : string = '.cfg'`

`ConfigExtension` is the default extension used by the `GetAppConfigFile (1460)` call. It can be set to any valid extension for the current OS.

`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 = $0000003f
```

Use this attribute in the `FindFirst` (1441) call to find all matching files.

```
faArchive = $00000020
```

Attribute of a file, meaning the file has the archive bit set. Used in `TSearchRec` (1381) and `FindFirst` (1441)

```
faDirectory = $00000010
```

Attribute of a file, meaning the file is a directory. Used in `TSearchRec` (1381) and `FindFirst` (1441)

```
faHidden = $00000002
```

Attribute of a file, meaning the file is read-only. Used in `TSearchRec` (1381) and `FindFirst` (1441)

```
faReadOnly = $00000001
```

Attribute of a file, meaning the file is read-only. Used in `TSearchRec` (1381) and `FindFirst` (1441)

```
faSymLink = $00000040
```

`faSymLink` means the file (as returned e.g. by `FindFirst` (1441)/`FindNext` (1442)), is a symlink. It's ignored under Windows.

```
faSysFile = $00000004
```

Attribute of a file, meaning the file is a system file. Used in `TSearchRec` (1381) and `FindFirst` (1441)

```
faVolumeId = $00000008
```

Attribute of a file, meaning the entry contains the volume ID. Used in `TSearchRec` (1381) and `FindFirst` (1441)

```
feInvalidHandle : THandle = (-1)
```

`feInvalidHandle` is the return value of `FileOpen` (1436) in case of an error.

```
fmOpenRead = $0000
```

`fmOpenRead` is used in the `FileOpen` (1436) call to open a file in read-only mode.

```
fmOpenReadWrite = $0002
```

`fmOpenReadWrite` is used in the `FileOpen` (1436) call to open a file in read-write mode.

```
fmOpenWrite = $0001
```

`fmOpenWrite` is used in the `FileOpen` (1436) call to open a file in write-only mode.

```
fmShareCompat = $0000
```

`fmOpenShareCompat` is used in the `FileOpen` (1436) call OR-ed together with one of `fmOpenReadWrite` (1370), `fmOpenRead` (1370) or `fmOpenWrite` (1370), to open a file in a sharing modulus that is equivalent to sharing implemented in MS-DOS.

```
fmShareDenyNone = $0040
```

`fmShareDenyNone` is used in the `FileOpen` (1436) call OR-ed together with one of `fmOpenReadWrite` (1370), `fmOpenRead` (1370) or `fmOpenWrite` (1370), to open a file so other processes can read/write the file as well.

```
fmShareDenyRead = $0030
```

`fmOpenShareRead` is used in the `FileOpen` (1436) call OR-ed together with one of `fmOpenReadWrite` (1370), `fmOpenRead` (1370) or `fmOpenWrite` (1370), 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
```

`fmOpenShareWrite` is used in the `FileOpen` (1436) call OR-ed together with one of `fmOpenReadWrite` (1370), `fmOpenRead` (1370) or `fmOpenWrite` (1370), to open a file so other processes cannot write to it, they can only read.

```
fmShareExclusive = $0010
```

`fmOpenShareExclusive` is used in the `FileOpen` (1436) call OR-ed together with one of `fmOpenReadWrite` (1370), `fmOpenRead` (1370) or `fmOpenWrite` (1370), to open a file exclusively.

```
fsFromBeginning = 0
```

`fsFromBeginning` is used to indicate in the `FileSeek` (1438) call that a seek operation should be started at the start of the file.

```
fsFromCurrent = 1
```

`fsFromBeginning` is used to indicate in the `FileSeek` (1438) 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` (1438) 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.

`JulianEpoch = (-2415018.5)`

Starting point of the Julian calendar

`LeadBytes : Set of Char = []`

`LeadBytes` contains the set of bytes that serve as lead byte in a MBCS string.

`MaxCurrency : Currency = 922337203685477.0000`

Maximum currency value

`MaxDateTime : TDateTime = 2958465.99999`

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 = -922337203685477.0000`

Minimum Currency value

`MinDateTime : TDateTime = -693593.0`

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,

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

SecsPerMin = 60

Number of seconds per minute

SwitchChars = ['-']

The characters in this set will be used by the FindCmdLineSwitch (1441) 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 (1459) 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 = (UnixEpoch)

Number of days between 1.1.1900 and 1.1.1970

UnixEpoch = JulianEpoch + (2440587.5)

Starting point of the unix calendar (1/1/1970)

37.13.2 Types

`EHeapException = EHeapMemoryError`

`EHeapMemoryError` is raised when an error occurs in the heap management routines.

`ExceptClass = Class of Exception`

`ExceptClass` is a `Exception` ([1527](#)) class reference.

`Int128Rec = packed record`
`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`
`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`
`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 of High and Low parts are different on various CPUs.

`OWordRec = packed record`
`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` ([1376](#)). Use to access memory regions as a byte array.

`PDayTable = ^TDayTable`

Pointer to `TDayTable` type.

`PString = ^string`

Pointer to a `ansistring`

`PSysCharSet = ^TSysCharSet`

Pointer to `TSysCharSet` (1381) type.

`PWordarray = ^TWordArray`

Generic pointer to `TWordArray` (1382). Use to access memory regions as a word array.

`TBeepHandler = procedure`

`TBeepHandler` is the prototype used by the `OnBeep` (1385) handler. This in turn is called by the `Beep` (1403) 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` (1375) type.

`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.

`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.

`TEventType = (etCustom, etInfo, etWarning, etError, etDebug)`

Table 37.10: Enumeration values for type `TEventType`

Value	Explanation
<code>etCustom</code>	Custom log event, with application-specific meaning
<code>etDebug</code>	Debug message.
<code>etError</code>	Error condition message
<code>etInfo</code>	General information event message
<code>etWarning</code>	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 37.11: Enumeration values for type

Value	Explanation
ExecInheritsHandles	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 (1425) call.

TFilename = String

TFileName is used in the TSearchRec (1381) definition.

TFileRec = FileRec

Alias for FileRec (1360) for Delphi compatibility.

TFileSearchOption = (sfoImplicitCurrentDir, sfoStripQuotes)

Table 37.12: 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 (1377) values, used in the FileSearch (1437) call when searching for files.

TFloatFormat = (ffGeneral, ffExponent, ffFixed, ffNumber, ffCurrency)

Table 37.13: 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 (1447) function.

```
TFloatRec = record
  Exponent : Integer;
  Negative : Boolean;
  Digits : Array[0..18] of Char;
end
```

TFloatRec is used to describe a floating point value by the FloatToDecimal (1443) function.

```
TFloatValue = (fvExtended, fvCurrency, fvSingle, fvReal, fvDouble, fvComp)
```

Table 37.14: 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 (1506) function.

```
TFormatDateTimeOption = (fdoInterval)
```

Table 37.15: 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 (1457) routine.

```
TFormatDateTimeOptions = Set of TFormatDateTimeOption
```

TFormatDateTimeOptions is a set of TFormatDateTimeOptions (1378), and is used in the last argument of FormatDateTime (1457)

```
TFormatSettings = record
  CurrencyFormat : Byte;
  NegCurrFormat : Byte;
  ThousandSeparator : Char;
  DecimalSeparator : Char;
  CurrencyDecimals : Byte;
```

```

DateSeparator : Char;
TimeSeparator : Char;
ListSeparator : Char;
CurrencyString : string;
ShortDateFormat : string;
LongDateFormat : string;
TimeAMString : string;
TimePMString : string;
ShortTimeFormat : string;
LongTimeFormat : string;
ShortMonthNames : TMonthNameArray;
LongMonthNames : TMonthNameArray;
ShortDayNames : TWeekNameArray;
LongDayNames : TWeekNameArray;
TwoDigitYearCenturyWindow : Word;
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.

```
TGetAppNameEvent = function : string
```

This callback type is used by the OnGetApplicationName ([1385](#)) to return an alternative application name.

```
TGetTempDirEvent = function(Global: Boolean) : string
```

Function prototype for OnGetTempDir ([1385](#)) handler.

```
TGetTempFileEvent = function(const Dir: string;const Prefix: string)
                        : string
```

Function prototype for OnGetTempFile ([1385](#)) handler.

```
TGetVendorNameEvent = function : string
```

TGetVendorNameEvent is the function prototype for the OnGetVendorName ([1385](#)) callback, used by the VendorName ([1516](#)) function.

```
THandle = System.THandle
```

THandle refers to the definition of THandle in the system unit, and is provided for backward compatibility only.

```
TIntegerSet = Set of
```

TIntegerSet is a generic integer subrange set definition whose size fits in a single integer.

```
TMbcsByteType = (mbSingleByte,mbLeadByte,mbTrailByte)
```


Table 37.16: Enumeration values for type `TMbcsByteType`

Value	Explanation
<code>mbLeadByte</code>	Uses lead-byte
<code>mbSingleByte</code>	Single bytes
<code>mbTrailByte</code>	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.

```
TProcedure = procedure
```

`TProcedure` is a general definition of a procedural callback.

```
TRawbyteSearchRec = record
  Time : LongInt;
  Size : Int64;
  Attr : LongInt;
  Name : RawByteString;
  ExcludeAttr : LongInt;
  FindHandle : THandle;
  Mode : TMode;
end
```

`TRawbyteSearchRec` is a search handle description record using single-byte strings. It is initialized by a call to `FindFirst` (1441) and can be used to do subsequent calls to `FindNext` (1442). It contains the result of these function calls. It must be used to close the search sequence with a call to `FindClose` (1440).

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.

```
TReplaceFlags= Set of (rfReplaceAll,rfIgnoreCase)
```

Table 37.17: Enumeration values for type

Value	Explanation
<code>rfIgnoreCase</code>	Search case insensitive.
<code>rfReplaceAll</code>	Replace all occurrences of the search string with the replacement string.

`TReplaceFlags` determines the behaviour of the `StringReplace` (1487) function.

TSearchRec = TRawbyteSearchRec

TSearchRec is a search handle description record. It is initialized by a call to FindFirst (1441) and can be used to do subsequent calls to FindNext (1442). It contains the result of these function calls. It must be used to close the search sequence with a call to FindClose (1440).

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)

TSignalState = (ssNotHooked, ssHooked, ssOverridden)

Table 37.18: 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.

TSysCharSet = Set of AnsiChar

Generic set of characters type.

```
TSysLocale = record
  DefaultLCID : Integer;
  PriLangID : Integer;
  SubLangID : Integer;
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.

TTerminateProc = function : Boolean

TTerminateProc is the procedural type which should be used when adding exit procedures.

TTextRec = TextRec

Alias for TextRec (1360) for Delphi compatibility.

```
TTimeStamp = record
  Time : LongInt;
  Date : LongInt;
end
```

TTimeStamp contains a timestamp, with the date and time parts specified as separate TDateTime values.

```
TUnicodeSearchRec = record
  Time : LongInt;
  Size : Int64;
  Attr : LongInt;
  Name : UnicodeString;
  ExcludeAttr : LongInt;
  FindHandle : THandle;
  Mode : TMode;
end
```

TRawbyteSearchRec is a search handle description record using multi-byte strings. It is initialized by a call to FindFirst (1441) and can be used to do subsequent calls to FindNext (1442). It contains the result of these function calls. It must be used to close the search sequence with a call to FindClose (1440).

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)

```
TWeekNameArray = Array[1..7] of string
```

TWeekNameArray is used in the day long and short name arrays.

```
TWordArray = Array[0..16383] of Word
```

TWordArray is a generic array definition, mostly for use as a base type of the PWordArray (1376) type.

```
WordRec = packed record
  Hi : Byte;
  Lo : 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.

37.13.3 Variables

```
CurrencyDecimals : Bytedeprecated
```

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.

```
CurrencyFormat : Bytedeprecated
```

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.

```
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.

`DateSeparator` : `Char` deprecated

`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.

`DecimalSeparator` : `Char` deprecated

`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.

`DefaultFormatSettings` : `TFormatSettings` = (`CurrencyFormat`: 1; `NegCurrFormat`: 5; `Thou`

`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` (1495).

`FormatSettings` : `TFormatSettings`

`FormatSettings` is provided for Delphi compatibility, and refers to the `DefaultFormatSettings` (1383) variable.

`ListSeparator` : `Char` 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.

`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.

`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.

`LongTimeFormat` : `stringdeprecated`

`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.

`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` (1410) 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` (1459) and `GetAppConfigFile` (1460) 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` (1464) 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` (1465) function. Note that the values for `Prefix` and `Dir` should be observed.

`OnGetVendorName` : `TGetVendorNameEvent`

`OnGetVendorName` must be set in order for `VendorName` (1516) to return a value. It will then be used in `GetAppConfigDir` (1459) and `GetAppConfigFile` (1460) 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` (1480) 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.

`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.

`ShortMonthNames` : `TMonthNameArray`~~deprecated~~

`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.

`ShortTimeFormat` : `string`~~deprecated~~

`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.

`SysLocale` : `TSysLocale`

`SysLocale` is initialized by the initialization code of the `SysUtils` unit. For an explanation of the fields, see `TSysLocale` ([1381](#))

`ThousandSeparator` : `Char`~~deprecated~~

`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.

`TimeAMString` : `string`~~deprecated~~

`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.

`TimePMString` : `string`~~deprecated~~

`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.

`TimeSeparator` : `Char`~~deprecated~~

`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` ([1495](#)).

`TwoDigitYearCenturyWindow` : `Word`

Window to determine what century 2 digit years are in.

37.14 Procedures and functions

37.14.1 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.

37.14.2 Abort

Synopsis: Abort program execution.

Declaration: `procedure Abort`

Visibility: default

Description: `Abort` raises an `EAbort` (1520) exception.

See also: `EAbort` (1520)

37.14.3 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` (1419) and `DiskSize` (1419) 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` (1419), `DiskSize` (1419)

37.14.4 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` ([1381](#)), `CallTerminateProcs` ([1405](#))

37.14.5 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. This is `#13#10` on Windows and Dos. On Unix-like platforms, this is `#10` and for Mac OS X it is `#13`.

Errors: None.

See also: `AnsiCompareStr` ([1389](#)), `AnsiCompareText` ([1390](#))

Listing: `./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.

37.14.6 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` (1389) or `AnsiCompareText` (1390) depending on whether `FileNameCaseSensitive` is `True` or `False`

Errors: None.

See also: `AnsiCompareStr` (1389), `AnsiCompareText` (1390), `AnsiLowerCaseFileName` (1393)

37.14.7 AnsiCompareStr

Synopsis: Compare 2 ansistrings, case sensitive, ignoring accents characters.

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 takes care of strange accented characters. Contrary to `AnsiCompareText` (1390), 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` (1388), `AnsiCompareText` (1390)

Listing: `./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.

```

37.14.8 AnsiCompareText

Synopsis: Compare 2 ansistrings, case insensitive, ignoring accents characters.

Declaration: `function AnsiCompareText(const S1: string;const S2: string) : Integer`

Visibility: default

Description: `AnsiCompareText` compares two strings and returns the following result:

```

<0if S1<S2.
0if S1=S2.
>0if S1>S2.

```

the comparison takes into account Ansi characters, i.e. it takes care of strange accented characters. Contrary to `AnsiCompareStr` ([1389](#)), 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` ([1388](#)), `AnsiCompareText` ([1390](#))

Listing: `./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.

```

37.14.9 AnsiDequotedStr

Synopsis:

Declaration: `function AnsiDequotedStr(const S: string; AQuote: Char) : string`

Visibility: default

Description:

37.14.10 AnsiExtractQuotedStr

Synopsis: Removes the first quoted string from a string.

Declaration: `function AnsiExtractQuotedStr(var Src: PChar; Quote: Char) : string`

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` ([1393](#)).

Errors: None.

See also: `AnsiQuotedStr` ([1393](#))

Listing: `./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.

```

37.14.11 AnsiLastChar

Synopsis: Return a pointer to the last character of a string.

Declaration: `function AnsiLastChar(const S: string) : PChar`

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` ([1396](#))

Listing: `./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.

37.14.12 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` ([1400](#)), `AnsiStrLower` ([1398](#)), `AnsiStrUpper` ([1400](#))

Listing: `./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.
```

37.14.13 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 \(1392\)](#), [AnsiCompareFileName \(1388\)](#), [AnsiUpperCaseFileName \(1401\)](#)

37.14.14 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 \(1399\)](#), [AnsiStrScan \(1399\)](#), [AnsiStrRScan \(1399\)](#)

37.14.15 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 \(1391\)](#).

For an example, see [AnsiExtractQuotedStr \(1391\)](#)

Errors: None.

See also: [AnsiExtractQuotedStr \(1391\)](#)

37.14.16 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` (1389) 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` (1389), `SameText` (1479), `AnsiSameText` (1394)

37.14.17 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` (1390) with `S1` and `S2` as parameters and returns `True` if the result of that call is zero, or `False` otherwise.

See also: `AnsiCompareText` (1390), `SameText` (1479), `AnsiSameStr` (1394)

37.14.18 AnsiStrComp

Synopsis: Compare two null-terminated strings. Case sensitive.

Declaration: `function AnsiStrComp(S1: PChar;S2: PChar) : 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` (1390), `AnsiCompareStr` (1389)

Listing: `./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.

```

37.14.19 AnsiStrIComp

Synopsis: Compare two null-terminated strings. Case insensitive.

Declaration: `function AnsiStrIComp(S1: PChar;S2: PChar) : Integer`

Visibility: default

Description: `AnsiStrIComp` compares 2 `PChar` strings, and returns the following result:

```

<0if S1<S2.
0if S1=S2.
>0if 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` ([1390](#)), `AnsiCompareStr` ([1389](#))

Listing: `./sysutex/ex55.pp`

```

Program Example55;

{ This program demonstrates the AnsiStrIComp function }

```

```

Uses sysutils;

Procedure TestIt (S1,S2 : Pchar);

Var R : Longint;

begin
  R:=AnsiStrlComp(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.

```

37.14.20 AnsiStrLastChar

Synopsis: Return a pointer to the last character of a string.

Declaration: `function AnsiStrLastChar(Str: PChar) : PChar`

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 \(1390\)](#), [AnsiCompareStr \(1389\)](#)

Listing: ./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 ' );
  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.

```

37.14.21 AnsiStrLComp

Synopsis: Compare a limited number of characters of 2 strings

Declaration: `function AnsiStrLComp(S1: PChar;S2: PChar;MaxLen: Cardinal) : Integer`

Visibility: default

Description: `AnsiStrLComp` functions the same as `AnsiStrComp` ([1394](#)), 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` ([1394](#)), `AnsiStrIComp` ([1395](#)), `AnsiStrLComp` ([1397](#))

37.14.22 AnsiStrLIComp

Synopsis: Compares a given number of characters of a string, case insensitive.

Declaration: `function AnsiStrLIComp(S1: PChar;S2: PChar;MaxLen: Cardinal) : Integer`

Visibility: default

Description: `AnsiStrLIComp` compares the first `Maxlen` characters of 2 `PChar` strings, `S1` and `S2`, and returns the following result:

<0if `S1<S2`.

0if `S1=S2`.

>0if `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` ([1390](#)), `AnsiCompareStr` ([1389](#))

Listing: `./sysutex/ex57.pp`

```

Program Example57;

{ 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 ');
  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.

```

37.14.23 AnsiStrLower

Synopsis: Convert a null-terminated string to all-lowercase characters.

Declaration: `function AnsiStrLower(Str: PChar) : PChar`

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` (1392). 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` (1400), `AnsiLowerCase` (1392)

Listing: `./sysutex/ex59.pp`

```

Program Example59;

{ This program demonstrates the AnsiStrLower function }

Uses sysutils;

```

```

Procedure Testit (S : PChar);

begin
  WriteLn (S, ' -> ', AnsiStrLower(S))
end;

Begin
  Testit('AN UPPERCASE STRING');
  Testit('Some mixed STring');
  Testit('a lowercase string');
End.

```

37.14.24 AnsiStrPos

Synopsis: Return position of one null-terminated substring in another

Declaration: `function AnsiStrPos(str: PChar; substr: PChar) : PChar`

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` ([1393](#)), `AnsiStrScan` ([1399](#)), `AnsiStrRScan` ([1399](#))

37.14.25 AnsiStrRScan

Synopsis: Find last occurrence of a character in a null-terminated string.

Declaration: `function AnsiStrRScan(Str: PChar; Chr: Char) : PChar`

Visibility: default

Description: `AnsiStrRScan` 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` ([1393](#)), `AnsiStrScan` ([1399](#)), `AnsiStrPos` ([1399](#))

37.14.26 AnsiStrScan

Synopsis: Find first occurrence of a character in a null-terminated string.

Declaration: `function AnsiStrScan(Str: PChar; Chr: Char) : PChar`

Visibility: default

Description: `AnsiStrScan` 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` ([1393](#)), `AnsiStrScan` ([1399](#)), `AnsiStrPos` ([1399](#))

37.14.27 AnsiStrUpper

Synopsis: Convert a null-terminated string to all-uppercase characters.

Declaration: `function AnsiStrUpper(Str: PChar) : PChar`

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` (1400). 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` (1400), `AnsiStrLower` (1398), `AnsiLowerCase` (1392)

Listing: `./sysutex/ex60.pp`

Program Example60;

{ This program demonstrates the AnsiStrUpper function }

Uses sysutils;

Procedure Testit (S : Pchar);

begin

 WriteLn (S, ' -> ', AnsiStrUpper(S))

end;

Begin

 Testit('AN UPPERCASE STRING');

 Testit('Some mixed STring');

 Testit('a lowercase string');

End.

37.14.28 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` (1400), `AnsiStrLower` (1398), `AnsiLowerCase` (1392)

Listing: `./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.

```

37.14.29 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` ([1400](#)), `AnsiCompareFileName` ([1388](#)), `AnsiLowerCaseFileName` ([1393](#))

37.14.30 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` ([1402](#)), `NewStr` ([1475](#)), `DisposeStr` ([1420](#))

Listing: `./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.

```

37.14.31 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` (1385) 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` (1459), `OnGetApplicationName` (1385), `GetAppConfigFile` (1460), `ConfigExtension` (1368)

37.14.32 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` (1475), `AppendStr` (1401), `DisposeStr` (1420)

Listing: `./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.

```

37.14.33 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` ([1500](#)), `IntToStr` ([1470](#))

Listing: `./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.

```

37.14.34 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` ([1385](#)) callback. The `Sysutils` unit itself contains no implementation of this call.

37.14.35 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 one of the strings `' TRUE'` or `' FALSE'`

Errors: None.

See also: `StrToBool` ([1495](#))

37.14.36 BytesOf

Synopsis: Return the bytes in a string

Declaration: `function BytesOf(const Val: RawByteString) : TBytes`
`function BytesOf(const Val: AnsiChar) : TBytes`

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` ([1376](#))

37.14.37 ByteToCharIndex

Synopsis: Convert a character index in Bytes to an Index in characters

Declaration: `function ByteToCharIndex(const S: string; Index: Integer) : Integer`

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` ([1406](#)), `ByteToCharLen` ([1404](#))

37.14.38 ByteToCharLen

Synopsis: Convert a length in bytes to a length in characters.

Declaration: `function ByteToCharLen(const S: string; MaxLen: Integer) : Integer`

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` ([1406](#)), `ByteToCharIndex` ([1404](#))

37.14.39 ByteType

Synopsis: Return the type of byte in an ansistring for a multi-byte character set

Declaration: `function ByteType(const S: string; Index: Integer) : 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` ([1380](#)), `StrByteType` ([1483](#))

37.14.40 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` ([1388](#)), 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` ([1381](#)), `AddTerminateProc` ([1388](#))

37.14.41 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.

Errors: None.

See also: `ExtractFileExt` ([1429](#)), `ExtractFileName` ([1429](#)), `ExtractFilePath` ([1429](#)), `ExpandFileName` ([1426](#))

37.14.42 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.

37.14.43 CharToByteLen

Synopsis: Convert a length in characters to a length in bytes.

Declaration: `function CharToByteLen(const S: string;MaxLen: Integer) : Integer`

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` ([1404](#)), `ByteToCharIndex` ([1404](#))

37.14.44 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` ([1406](#))

37.14.45 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` ([1406](#))

37.14.46 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.

37.14.47 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` ([1479](#))

37.14.48 CompareStr

Synopsis: Compare 2 ansistrings case-sensitively, ignoring special characters.

Declaration: `function CompareStr(const S1: string;const S2: string) : Integer
; Overload`

Visibility: default

Description: `CompareStr` 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-sensitive. The function does not take internationalization settings into account, it simply compares ASCII values.

Errors: None.

See also: `AnsiCompareText` ([1390](#)), `AnsiCompareStr` ([1389](#)), `CompareText` ([1408](#))

Listing: `./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 ' );
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.

```

37.14.49 CompareText

Synopsis: Compare 2 ansistrings case insensitive.

Declaration: `function CompareText(const S1: string;const S2: string) : Integer`

Visibility: default

Description: CompareText compares two strings, S1 and S2, and returns the following result:

```

<0if S1<S2.
0if S1=S2.
>0if 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` ([1390](#)), `AnsiCompareStr` ([1389](#)), `CompareStr` ([1407](#))

Listing: ./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

```

```
// 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.
```

37.14.52 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` ([1477](#))

Listing: `./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.

37.14.53 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` ([1466](#)), `IsEqualGUID` ([1471](#)), `StringToGUID` ([1488](#)), `TryStringToGUID` ([1512](#)), `GUIDToString` ([1466](#))

37.14.54 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` ([1412](#)), `Time` ([1507](#)), `Now` ([1475](#))

37.14.55 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` ([1445](#)):

```
FloatToStrF(Value, ffNumber, 15, 2);
```

Errors: None.

See also: `FloatToStrF` ([1445](#)), `StrToCurr` ([1496](#))

37.14.56 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` ([1445](#)).

See also: `FloatToStrF` ([1445](#))

37.14.57 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` ([1507](#)), `Now` ([1475](#))

Listing: `./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.

37.14.58 DateTimeToFileDate

Synopsis: Convert a `TDateTime` value to a file age (integer)

Declaration: `function DateTimeToFileDate(DateTime: TDateTime) : LongInt`

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` ([1431](#)) function.

Errors: None.

See also: `Time` ([1507](#)), `Date` ([1412](#)), `FileDateToDateTime` ([1433](#)), `DateTimeToSystemTime` ([1414](#)), `DateTimeToTimeStamp` ([1415](#))

Listing: `./sysutex/ex2.pp`

Program `Example2`;

{ This program demonstrates the DateTimeToFileDate function }

Uses `sysutils`;

Begin

`WriteLn ('FileTime of now would be: ',DateTimeToFileDate (Now));`

End.

37.14.59 DateTimeToStr

Synopsis: Converts a TDateTime value to a string using a predefined format.

Declaration: `function DateTimeToStr(DateTime: TDateTime) : string`
`function DateTimeToStr(DateTime: TDateTime;`
`const FormatSettings: TFormatSettings) : string`

Visibility: default

Description: `DateTimeToStr` returns a string representation of `DateTime` using the formatting specified in `LongDateTimeFormat`. It corresponds to a call to `FormatDateTime('c', DateTime)` (see [formatchars \(1366\)](#)).

Errors: None.

See also: `FormatDateTime` ([1457](#))

Listing: `./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.

37.14.60 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 \(1366\)](#).

Errors: In case a wrong formatting character is found, an `EConvertError` is raised.

See also: `FormatDateTime` ([1457](#)), [formatchars \(1366\)](#)

Listing: `./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 mmm y');

 Today ('Today is "d mm yy');

 Today ('Today is "d/mm/yy');

Now ('The time is 'am/pmh:n:s');

Now ('The time is 'hh:nn:ssam/pm');

Now ('The time is 'tt');

End.

37.14.61 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 ([1412](#)), SystemTimeToDateTime ([1505](#)), DateTimeToTimeStamp ([1415](#))

Listing: ./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.

```

37.14.62 DateTimeToTimeStamp

Synopsis: Converts a TDateTime value to a TimeStamp structure.

Declaration: function DateTimeToTimeStamp(DateTime: TDateTime) : TTimeStamp

Visibility: default

Description: DateTimeToSystemTime converts a date/time pair in DateTime, with TDateTime format to a TTimeStamp format.

Errors: None.

See also: DateTimeToFileDate ([1412](#)), SystemTimeToDateTime ([1505](#)), DateTimeToSystemTime ([1414](#))

Listing: ./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.

```

37.14.63 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)`.

Errors: None.

See also: `TimeToStr` ([1509](#)), `DateTimeToStr` ([1413](#)), `FormatDateTime` ([1457](#)), `StrToDate` ([1497](#))

Listing: `./sysutex/ex7.pp`

Program `Example7`;

{ This program demonstrates the DateToStr function }

Uses `sysutils`;

Begin

`WriteLn(Format('Today is: %s',[DateToStr(Date)]));`
End.

37.14.64 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` ([1412](#)), `DateToStr` ([1415](#))

Listing: `./sysutex/ex8.pp`

Program `Example8`;

{ This program demonstrates the DayOfWeek function }

Uses `sysutils`;

Begin

`WriteLn('Today's day is ',LongDayNames[DayOfWeek(Date)]);`
End.

37.14.65 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` ([1421](#)), `DecodeTime` ([1417](#))

Listing: `./sysutex/ex9.pp`

Program `Example9`;

{ This program demonstrates the DecodeDate function }

Uses `sysutils`;

Var `YY,MM,DD` : `Word`;

Begin

`DecodeDate (Date ,YY,MM,DD);`

`WriteIn (Format ('Today is %d/%d/%d' ,[dd,mm,yy]));`

End.

37.14.66 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` ([1416](#)), 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` ([1421](#)), `TryEncodeDate` ([1511](#)), `DecodeDate` ([1416](#))

37.14.67 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: `DecodeDate` 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` ([1422](#)), `DecodeDate` ([1416](#))

Listing: `./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.

37.14.68 DeleteFile

Synopsis: Delete a file from the filesystem.

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 ([1432](#)), FileExists ([1433](#))

Listing: ./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[I],**Length**(Line));

FileClose(F);

DeleteFile('test.txt');

End.

37.14.69 DirectoryExists

Synopsis: Check whether a directory exists in the file system.

Declaration: function DirectoryExists(const Directory: UnicodeString) : Boolean
function DirectoryExists(const Directory: RawByteString) : Boolean

Visibility: default

Description: `DirectoryExists` checks whether `Directory` exists in the filesystem and is actually a directory. If this is the case, the function returns `True`, otherwise `False` is returned.

See also: `FileExists` ([1433](#))

37.14.70 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 filesystem is seen as one big directory tree. For this reason, the `DiskFree` and `DiskSize` ([1419](#)) functions must be mimicked using filenames that reside on the partitions. For more information, see `AddDisk` ([1387](#)).

Errors: On error, `-1` is returned.

See also: `DiskSize` ([1419](#)), `AddDisk` ([1387](#))

Listing: `./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.

37.14.71 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 filesystem is seen as one big directory tree. For this reason, the `DiskFree` (1419) and `DiskSize` functions must be mimicked using filenames that reside on the partitions. For more information, see `AddDisk` (1387)

For an example, see `DiskFree` (1419).

Errors: On error, `-1` is returned.

See also: `DiskFree` (1419), `AddDisk` (1387)

37.14.72 DisposeStr

Synopsis: Dispose an anstring 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` (1420).

Errors: None.

See also: `NewStr` (1475), `AppendStr` (1401), `AssignStr` (1402)

37.14.73 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` (1124) constant.

Errors: None.

See also: `ExtractFileName` (1429), `ExtractFilePath` (1429)

Listing: `./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.
```

37.14.74 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 (1416) procedure.

The parameters must lie within 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 its valid range, an EConvertError (1521) exception is raised.

See also: EncodeTime (1422), DecodeDate (1416)

Listing: ./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 mmm yyyy ',EncodeDate(YY,Mm,Dd)));
```

```
End.
```

37.14.75 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` (1417) 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` (1521) exception is raised.

See also: `EncodeDate` (1421), `DecodeTime` (1417)

Listing: ./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.

37.14.76 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` (1423), `ExceptionErrorMessage` (1423), `ShowException` (1480)

37.14.77 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` (1423), `ExceptAddr` (1422), `ExceptObject` (1423), `#rtl.system.ExceptProc` (1126)

37.14.78 ExceptFrames

Synopsis:

Declaration: `function ExceptFrames : PCodePointer`

Visibility: default

Description:

See also: `ExceptFrameCount` (1422), `ExceptAddr` (1422), `ExceptObject` (1423), `#rtl.system.ExceptProc` (1126)

37.14.79 ExceptionErrorMessage

Synopsis: Return a message describing the exception.

Declaration: `function ExceptionErrorMessage(ExceptObject: TObject;
ExceptAddr: Pointer; Buffer: PChar;
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` (1527) object or not, and adapts the output accordingly.

See also: `ExceptObject` (1423), `ExceptAddr` (1422), `ShowException` (1480)

37.14.80 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` (1422), `ExceptionErrorMessage` (1423), `ShowException` (1480)

37.14.81 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 \(1468\)](#), [IncludeLeadingPathDelimiter \(1467\)](#), [ExcludeTrailingPathDelimiter \(1424\)](#), [ConcatPaths \(1409\)](#)

Listing: ./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.

```

37.14.82 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` ([1424](#)) instead.

See also: IncludeTrailingPathDelimiter (1468), ExcludeTrailingPathDelimiter (1424), PathDelim (1372), IsPathDelimiter (1472)

37.14.83 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 (`PathDelim` ([1372](#))) from `Path` if it is present, and returns the result.

See also: `ExcludeTrailingBackslash` ([1424](#)), `IncludeTrailingPathDelimiter` ([1468](#)), `PathDelim` ([1372](#)), `IsPathDelimiter` ([1472](#))

37.14.84 `ExecuteProcess`

Synopsis: Execute another process (program).

Declaration:

```
function ExecuteProcess(const Path: AnsiString;
                       const ComLine: AnsiString; Flags: TExecuteFlags)
                       : Integer
function ExecuteProcess(const Path: AnsiString;
                       const ComLine: Array of AnsiString;
                       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` ([1525](#)) exception will be raised.

See also: `TExecuteFlags` ([1377](#)), `EOSError` ([1525](#))

37.14.85 `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` ([1138](#))). 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` ([1437](#))

37.14.86 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.

Errors: None.

See also: `ExpandFileNameCase` (1426), `ExtractFileName` (1429), `ExtractFilePath` (1429), `ExtractFileDir` (1428), `ExtractFileDrive` (1428), `ExtractFileExt` (1429), `ExtractRelativePath` (1430)

Listing: ./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.

37.14.87 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` (1426) 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 (1441)/FindNext (1442).

Errors: None.

See also: [ExpandFileName \(1426\)](#), [ExtractFileName \(1429\)](#), [ExtractFilePath \(1429\)](#), [ExtractFileDir \(1428\)](#), [ExtractFileDrive \(1428\)](#), [ExtractFileExt \(1429\)](#), [ExtractRelativePath \(1430\)](#)

Listing: ./sysutex/ex33.pp

Program Example33;

```
{ This program demonstrates the ExpandFileName function }
```

Uses sysutils;

```
Procedure Testit (F : String);
```

begin

```

WriteIn (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 .

37.14.88 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 (1426) on FileName and then attempts to replace the drive letter by the name of a shared disk.

Errors: None.

See also: [ExpandFileName \(1426\)](#), [ExtractFileName \(1429\)](#), [ExtractFilePath \(1429\)](#), [ExtractFileDir \(1428\)](#), [ExtractFileDrive \(1428\)](#), [ExtractFileExt \(1429\)](#), [ExtractRelativePath \(1430\)](#)

37.14.89 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 driveletter. The directory name has NO ending directory separator, in difference with `ExtractFilePath` (1429).

Errors: None.

See also: `ExtractFileName` (1429), `ExtractFilePath` (1429), `ExtractFileDir` (1428), `ExtractFileDrive` (1428), `ExtractFileExt` (1429), `ExtractRelativePath` (1430)

Listing: ./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/mysqld');
Testit ('c:\pp\bin\win32\ppc386.exe');
Testit ('/pp/bin/win32/ppc386.exe');

End.

37.14.90 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` (1428).

See also: `ExtractFileName` (1429), `ExtractFilePath` (1429), `ExtractFileDir` (1428), `ExtractFileDrive` (1428), `ExtractFileExt` (1429), `ExtractRelativePath` (1430)

37.14.91 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` (1428).

Errors: None.

See also: `ChangeFileExt` (1405), `ExtractFileName` (1429), `ExtractFilePath` (1429), `ExtractFileDir` (1428), `ExtractFileDrive` (1428), `ExtractFileExt` (1429), `ExtractRelativePath` (1430)

37.14.92 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` (1429) and `ExtractFileName`.

For an example, see `ExtractFileDir` (1428).

Errors: None.

See also: `ExtractFileName` (1429), `ExtractFilePath` (1429), `ExtractFileDir` (1428), `ExtractFileDrive` (1428), `ExtractFileExt` (1429), `ExtractRelativePath` (1430)

37.14.93 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 driveletter) 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` (1429).

For an example, see `ExtractFileDir` (1428).

Errors: None.

See also: `ExtractFileName` (1429), `ExtractFilePath` (1429), `ExtractFileDir` (1428), `ExtractFileDrive` (1428), `ExtractFileExt` (1429), `ExtractRelativePath` (1430)

37.14.94 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: This function does not exist in the Delphi unit.

Errors: None.

See also: `ExtractFileName` (1429), `ExtractFilePath` (1429), `ExtractFileDir` (1428), `ExtractFileDrive` (1428), `ExtractFileExt` (1429)

Listing: ./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 ('e:/pp/bin/win32/ppc386', 'd:/pp/src/compiler');

 Testit ('e:\pp\bin\win32\ppc386', 'd:\pp\src\compiler');

End.

37.14.95 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` (1429), `ExtractFileName` (1429)

37.14.96 FileAge

Synopsis: Return the timestamp of a file.

Declaration: `function FileAge(const FileName: UnicodeString) : LongInt`
`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) : LongInt`

Visibility: default

Description: `FileAge` returns the last modification time of file `FileName`. The `FileDate` format can be transformed to `TDateTime` format with the `FileDateToDateTime` (1433) function.

Fileage cannot be used on directories, it will return -1 if `FileName` indicates a directory.

Errors: In case of errors, -1 is returned.

See also: `FileDateToDateTime` (1433), `FileExists` (1433), `FileGetAttr` (1434)

Listing: `./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.

37.14.97 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` (1432)

Errors: None.

See also: `FileCreate` (1432), `FileWrite` (1440), `FileOpen` (1436), `FileRead` (1437), `FileTruncate` (1440), `FileSeek` (1438)

37.14.98 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` (1437) and `FileWrite` (1440) functions.

If a file with name `FileName` already existed on the disk, it is overwritten.

The optional `Mode` 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 platfors, the `Mode` parameter is ignored.

Errors: If an error occurs (e.g. disk full or non-existent path), the function returns `THandle(-1)`.

See also: `FileClose` (1431), `FileWrite` (1440), `FileOpen` (1436), `FileRead` (1437), `FileTruncate` (1440), `FileSeek` (1438)

Listing: `./sysutex/ex37.pp`

Program Example37;

{ This program demonstrates the FileCreate function }

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);
  l:=50*SizeOf(Longint);
  If FileTruncate(F,l) then
    Writeln('Successfully truncated file to ',l,' bytes.');
```

```

  FileClose(F);
End.
```

37.14.99 FileDateToDateTime

Synopsis: Convert a `FileDate` value to a `TDateTime` value.

Declaration: `function FileDateToDateTime(Filedate: LongInt) : 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` (1431) or `FindFirst` (1441)/`FindNext` (1442) functions to `TDateTime` form.

Errors: None.

See also: `DateTimeToFileDate` (1412)

Listing: `./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.
```

37.14.100 FileExists

Synopsis: Check whether a particular file exists in the filesystem.

Declaration: `function FileExists(const FileName: UnicodeString) : Boolean`
`function FileExists(const FileName: RawByteString) : 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 unices, passing a directory name will result in `True`. The rationale is that on unix, a directory is a file as well.

Note that this function accepts a single filename as an argument, without wildcards. To check for the existence of multiple files, see the `FindFirst` (1441) function.

Errors: None.

See also: `FindFirst` (1441), `FileAge` (1431), `FileGetAttr` (1434), `FileSetAttr` (1439)

Listing: ./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.

37.14.101 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:

faReadOnlyThe file is read-only.

faHiddenThe file is hidden. (On unix, this means that the filename starts with a dot)

faSysFileThe file is a system file (On unix, this means that the file is a character, block or FIFO file).

faVolumeIdVolume Label. Only for DOS/Windows on a plain FAT (not VFAT or Fat32) filesystem.

faDirectoryFile is a directory.

faArchivefile should be archived. Not possible on Unix

Errors: In case of error, -1 is returned.

See also: `FileSetAttr` (1439), `FileAge` (1431), `FileGetDate` (1435)

Listing: ./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 artchive 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.

```

37.14.102 FileGetDate

Synopsis: Return the file time of an opened file.

Declaration: `function FileGetDate(Handle: THandle) : LongInt`

Visibility: default

Description: `FileGetdate` returns the filetype of the opened file with filehandle `Handle`. It is the same as `FileAge` ([1431](#)), 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` ([1431](#))

Listing: `./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**(**FileDateToDateTime**(D)));

```

FileClose(F);
DeleteFile('test.dat');
End.

```

37.14.103 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 filesystem and is a read-only file. If this is the case, the function returns `True`, otherwise `False` is returned.

See also: `FileExists` ([1433](#))

37.14.104 FileOpen

Synopsis: Open an existing file and return a filehandle

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:

fmOpenReadOnly Open file in read-only mode

fmOpenWriteOpen Open file in write-only mode

fmOpenReadWriteOpen Open 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 Open 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.

fmShareDenyNone Do 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` ([1437](#)) 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` ([1432](#)) 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` (1432)

Errors: On Error, `THandle (-1)` is returned.

See also: `fmOpenRead` (1370), `fmOpenWrite` (1370), `fmOpenReadWrite` (1370), `fmShareDenyWrite` (1370), `fmShareExclusive` (1370), `fmShareDenyRead` (1370), `fmShareDenyNone` (1370), `fmShareCompat` (1370), `FileClose` (1431), `FileWrite` (1440), `FileCreate` (1432), `FileRead` (1437), `FileTruncate` (1440), `FileSeek` (1438)

37.14.105 FileRead

Synopsis: Read data from a filehandle 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` (1436) call.

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

For an example, see `FileCreate` (1432)

Errors: On error, -1 is returned.

See also: `FileClose` (1431), `FileWrite` (1440), `FileCreate` (1432), `FileOpen` (1436), `FileTruncate` (1440), `FileSeek` (1438)

37.14.106 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` (1426), `FindFirst` (1441)

Listing: `./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.
```

37.14.107 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 abovementioned constants do not exist in Delphi.

Errors: On error, -1 is returned.

See also: [FileClose \(1431\)](#), [FileWrite \(1440\)](#), [FileCreate \(1432\)](#), [FileOpen \(1436\)](#), [FileRead \(1437\)](#), [FileTruncate \(1440\)](#)

Listing: ./sysutex/ex42.pp

Program Example42;

{ This program demonstrates the FileSetAttr function }

Uses sysutils;

Begin

```
  If FileSetAttr ( 'ex40.pp',faReadOnly or faHidden)=0 then
    Writeln ( 'Successfully made file hidden and read-only.')
  else
    Writeln ( 'Couldn't make file hidden and read-only.');
```

End.

37.14.108 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 pre-defined `faXXX` constants.

This function is not implemented on Unixes.

Errors: On error, -1 is returned (always on Unixes).

See also: [FileGetAttr \(1434\)](#), [FileGetDate \(1435\)](#), [FileSetDate \(1439\)](#)

37.14.109 FileSetDate

Synopsis: Set the date of a file.

Declaration: `function FileSetDate(const FileName: UnicodeString;Age: LongInt) : LongInt`
`function FileSetDate(const FileName: RawByteString;Age: LongInt) : LongInt`
`function FileSetDate(Handle: THandle;Age: LongInt) : 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 unices, 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.

37.14.110 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` (1432).

Errors: On error, the function returns `False`.

See also: `FileClose` (1431), `FileWrite` (1440), `FileCreate` (1432), `FileOpen` (1436), `FileRead` (1437), `FileSeek` (1438)

37.14.111 FileWrite

Synopsis: Write data from a buffer to a given filehandle.

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` (1432).

Errors: In case of error, -1 is returned.

See also: `FileClose` (1431), `FileCreate` (1432), `FileOpen` (1436), `FileRead` (1437), `FileTruncate` (1440), `FileSeek` (1438)

37.14.112 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` (1441)/`FindNext` (1442) 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` (1441).

Errors: None.

See also: `FindFirst` (1441), `FindNext` (1442)

37.14.113 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
```

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` (1374)) 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` (1374)

37.14.114 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:

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) filesystems.

faDirectory File is a directory.

faArchive file needs to be archived. Not possible on Unix

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.

The `Rslt` record can be fed to subsequent calls to `FindNext`, in order to find other files matching the specifications.

Remark: A `FindFirst` call must *always* be followed by a `FindClose` (1440) call with the same `Rslt` record. Failure to do so will result in memory loss.

Errors: On error the function returns -1 on Unix-like platforms, a nonzero error code on Windows.

See also: `FindClose` (1440), `FindNext` (1442)

Listing: ./sysutex/ex43.pp

Program Example43;

{ This program demonstrates the FindFirst function }

Uses SysUtils;

Var Info : TSearchRec;
Count : Longint;

Begin

Count:=0;

If FindFirst ('*',faAnyFile **and** faDirectory ,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;

end;

FindClose(Info);

WriteLn ('Finished search. Found ',Count,' matches');

End.

37.14.115 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` ([1441](#))

Errors: On error (no more file is found), a nonzero constant is returned.

See also: `FindFirst` ([1441](#)), `FindClose` ([1440](#))

37.14.116 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` ([1371](#)) and `MaxCurrency` ([1371](#))). If not, an `EConvertError` ([1521](#)) exception is raised.

Errors: If `Value` is out of range, an `EConvertError` ([1521](#)) exception is raised.

See also: `EConvertError` ([1521](#)), `TryFloatToCurr` ([1512](#)), `MinCurrency` ([1371](#)), `MaxCurrency` ([1371](#))

37.14.117 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` ([1371](#)) and `MaxDateTime` ([1371](#))). If not, an `EConvertError` ([1521](#)) exception is raised.

Errors: If `Value` is out of range, an `EConvertError` ([1521](#)) exception is raised.

See also: `EConvertError` ([1521](#)), `MinDateTime` ([1371](#)), `MaxDateTime` ([1371](#))

37.14.118 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` ([1378](#)) format. It will store `Precision` digits in the `Digits` field, of which at most `Decimal` decimals.

Errors: None.

See also: `TFloatRec` ([1378](#))

37.14.119 FloatToStr

Synopsis: Convert a float value to a string using a fixed format.

Declaration:

```
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` ([1445](#)). `FloatToStr` is completely equivalent to the following call:

```
FloatToStrF(Value, ffGeneral, 15, 0);
```

Errors: None.

See also: `FloatToStrF` ([1445](#)), `FormatFloat` ([1457](#)), `StrToFloat` ([1499](#))

Listing: `./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.

```

37.14.120 FloatToStrF

Synopsis: Convert a float value to a string using a given format.

Declaration:

```

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:

ffcurrencyMoney 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.

ffExponentScientific 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.

ffFixedFixed 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. 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.

ffnumber Is the same as `ffFixed`, except that thousand separators are inserted in the resulting string.

Errors: None.

See also: `FloatToStr` ([1444](#)), `FloatToText` ([1447](#))

Listing: ./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 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;

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.

37.14.121 FloatToText

Synopsis: Return a string representation of a float, with a given format.

Declaration: `function FloatToText (Buffer: PChar; Value: Extended; format: TFloatFormat;
Precision: Integer; Digits: Integer) : LongInt
function FloatToText (Buffer: PChar; 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` (1445). `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` (1444), `FloatToStrF` (1445)

Listing: ./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.

```

37.14.122 FloatToTextFmt

Synopsis: Convert a float value to a string using a given mask.

Declaration:

```

function FloatToTextFmt(Buffer: PChar;Value: Extended;format: PChar;
                        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` ([1457](#))

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` ([1457](#))

37.14.123 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` ([1449](#)) with `Fmt` and `Args` as arguments, and stores the result in `Res`. For more information on how the resulting string is composed, see `Format` ([1449](#)).

Errors: In case of error, a `EConvertError` exception is raised.

See also: `Format` ([1449](#)), `FormatBuf` ([1456](#))

Listing: `./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 fomattng see %s.', ['Format']);
```

```
  WriteLn (S);
```

```
End.
```

37.14.124 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 succesfully. If it failed to create any of the parts, `False` is returned.

37.14.125 Format

Synopsis: Format a string with given arguments.

Declaration: `function Format(const Fmt: string;const Args: Array of const) : string`
`function Format(const Fmt: string;const Args: Array of const;`
`const FormatSettings: TFormatSettings) : string`

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 `EConvertError` exception will be raised.

The argument type is determined from `ArgType`. It can have one of the following values (case insensitive):

DDecimal 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.

EScientific 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` (1445), 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)
```

FFixed 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` (1445)). `Precision` indicates the number of digits following the decimal point.

In short, the `F` specifier formats it's argument as follows:

```
FloatToStrF (Argument, ffFixed, fixed, 9999, Precision)
```

GGeneral 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)
```

MCurrency format. the next argument in the `var{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` (1445)

In short, the `M` specifier formats it's argument as follows:

```
FloatToStrF (Argument, ffCurrency, 9999, Precision)
```

NNumber format. This is the same as fixed point format, except that thousand separators are inserted in the resulting string.

PPointer 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.

SString format. The next argument in the `Args` array must be a string. The argument is simply copied to the result string. If `Precision` is specified, then only `Precision` characters are copied to the result string.

UUnsigned 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.

Xhexadecimal 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).

Errors: In case of error, an `EConversionError` 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` ([1456](#))

Listing: `./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
```

```
                WriteIn ( '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,[0.0123]); 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.

```

37.14.126 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` ([1449](#)) and stores the result in `Buf`.

See also: `Format` ([1449](#))

Listing: `./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.

37.14.127 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 `Format`, and returns the resulting string.

For an explanation of the formatting characters, see `FormatFloat` ([1457](#)).

See also: `FormatFloat` ([1457](#)), `FloatToText` ([1447](#))

37.14.128 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 \(1366\)](#).

Errors: On error (such as an invalid character in the formatting string), and `EConvertError` exception is raised.

See also: [DateTimeToStr \(1413\)](#), [DateToStr \(1415\)](#), [TimeToStr \(1509\)](#), [StrToDateTime \(1498\)](#)

Listing: `./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.

37.14.129 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).

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 (1445)` 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` ([1444](#))

Listing: `./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.

```

37.14.130 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.

37.14.131 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` (1402) call, but can be configured by means of the `OnGetApplicationName` (1385) callback.

If `VendorName` (1516) is not-empty, then `VendorName` will also be inserted before the application-specific directory.

Errors: None.

See also: `GetAppConfigFile` (1460), `ApplicationName` (1402), `OnGetApplicationName` (1385), `CreateDir` (1410), `SysConfigDir` (1374), `VendorName` (1516)

37.14.132 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` (1449) 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` (1402) call, but can be configured by means of the `OnGetApplicationName` (1385) callback.

If `VendorName` (1516) is not-empty, then `VendorName` will be inserted in the path for the config file directory.

Errors: None.

See also: `GetAppConfigDir` (1459), `OnGetApplicationName` (1385), `ApplicationName` (1402), `CreateDir` (1410), `ConfigExtension` (1368), `SysConfigDir` (1374), `VendorName` (1516)

37.14.133 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` (1480), `DiskFree` (1419), `DiskSize` (1419)

Listing: `./sysutex/ex28.pp`

Program Example28;

{ This program demonstrates the GetCurrentDir function }

Uses sysutils;

Begin

WriteLn ('Current Directory is : ',GetCurrentDir);

End.

37.14.134 GetDirs

Synopsis: Return a list of directory names from a path.

Declaration: function GetDirs(var DirName: UNICODESTRING;
 var Dirs: Array of PWideChar) : 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 ([1430](#))

Listing: ./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.

37.14.135 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` (1462).

For an example, `GetEnvironmentVariableCount` (1462).

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 proces. 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` (1462), `GetEnvironmentVariableCount` (1462)

37.14.136 GetEnvironmentVariable

Synopsis: Return the value of an environment variable.

Declaration: `function GetEnvironmentVariable(const EnvVar: AnsiString) : AnsiString`
`function GetEnvironmentVariable(const EnvVar: UnicodeString)`
`: UnicodeString`

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` (1462), `GetEnvironmentVariableCount` (1462)

37.14.137 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` (1462), `GetEnvironmentVariable` (1462)

Listing: `./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.

```

37.14.138 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.

37.14.139 GetLastOSError

Synopsis: Return the last code from the OS.

Declaration: `function GetLastOSError : Integer`

Visibility: default

Description: `GetLastOSError` 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, `GetLastOSError` 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: `RaiseLastOSError` ([1477](#))

37.14.140 GetLocalTime

Synopsis: Get the local time.

Declaration: `procedure GetLocalTime(var SystemTime: TSystemTime)`

Visibility: default

Description: `GetLocalTime` returns the system time in a `TSystemTime` ([1360](#)) format.

Errors: None.

See also: `Now` ([1475](#)), `Date` ([1412](#)), `Time` ([1507](#)), `TSystemTime` ([1360](#))

37.14.141 GetLocalTimeOffset

Synopsis: Return local timezone offset

Declaration: `function GetLocalTimeOffset : Integer`

Visibility: default

Description: `GetLocalTimeOffset` returns the local timezone offset in minutes. This is the difference between UTC time and local time:

```
UTC = LocalTime + GetLocalTimeOffset
```

Note that on Linux/Unix, this information may be inaccurate around the DST time changes (for optimization). In that case, the `unix.ReReadLocalTime` (1619) unit must be used to re-initialize the timezone information.

See also: `unix.ReReadLocalTime` (1619), `Date` (1412), `Time` (1507), `Now` (1475)

37.14.142 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.

37.14.143 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` (1385) 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` (1385), `GetTempFileName` (1465)

37.14.144 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: PChar;Prefix: PChar;uUnique: DWord;
TempFileName: PChar) : 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` ([1385](#)) 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` ([1464](#)), `OnGetTempFile` ([1385](#))

37.14.145 GetTickCount

Synopsis: Get tick count (32-bit, deprecated)

Declaration: `function GetTickCount : LongWord`

Visibility: default

Description: `GetTickCount` returns an increasing clock tick count. 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` ([1465](#)) instead.

See also: `GetTickCount64` ([1465](#)), `Now` ([1475](#)), `Time` ([1507](#)), `Sleep` ([1481](#))

37.14.146 GetTickCount64

Synopsis: Get tick count (64-bit)

Declaration: `function GetTickCount64 : QWord`

Visibility: default

Description: `GetTickCount64` returns an increasing clock tick count. It is useful for time measurements, but no assumptions should be made as to the interval between the ticks.

See also: `Now` ([1475](#)), `Time` ([1507](#)), `Sleep` ([1481](#))

37.14.147 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` ([1459](#))

37.14.148 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` ([1471](#))

37.14.149 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` ([1504](#)), `#rtl.system.TGUID` ([1166](#)), `StringToGUID` ([1488](#)), `IsEqualGuid` ([1471](#))

37.14.150 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.

37.14.151 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`.

37.14.152 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` (1468), but operates on an already decoded date.

See also: `IncMonth` (1468)

37.14.153 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` (1125)) 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` (1468), `ExcludeLeadingPathDelimiter` (1424), `ExcludeTrailingPathDelimiter` (1424), `ConcatPaths` (1409)

Listing: `./sysutex/ex94.pp`

Program `Example94`;

{ This program demonstrates the IncludeLeadingPathDelimiter function }

Uses `sysutils`;

Begin

End.

37.14.154 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` ([1468](#)) instead.

See also: `IncludeTrailingPathDelimiter` ([1468](#)), `ExcludeTrailingPathDelimiter` ([1424](#)), `PathDelim` ([1372](#)), `IsPathDelimiter` ([1472](#))

37.14.155 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` ([1372](#))) 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` ([1468](#)), `ExcludeTrailingPathDelimiter` ([1424](#)), `PathDelim` ([1372](#)), `IsPathDelimiter` ([1472](#))

37.14.156 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` ([1412](#)), `Time` ([1507](#)), `Now` ([1475](#))

Listing: `./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.

```

37.14.157 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.

ssHooked A signal handler is set by the RTL code for the signal.

ssOverridden A 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.

37.14.158 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`

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` ([1470](#))

Listing: ./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.

37.14.159 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 it's 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 ([1469](#)), StrToInt ([1500](#))

Listing: ./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.

37.14.160 IsDelimiter

Synopsis: Check whether a given string is a delimiter character.

Declaration: `function IsDelimiter(const Delimiters: string; const S: string;
Index: Integer) : 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` ([1473](#))

37.14.161 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` ([1504](#)), `#rtl.system.TGUID` ([1166](#)), `StringToGUID` ([1488](#)), `GuidToString` ([1466](#))

37.14.162 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` ([1468](#)), `Date` ([1412](#))

Listing: `./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.

```

37.14.163 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` ([1372](#)), i.e. if it is a path delimiter character for the current platform.

Errors: `IncludeTrailingPathDelimiter` ([1468](#))`ExcludeTrailingPathDelimiter` ([1424](#))`PathDelim` ([1372](#))

37.14.164 IsValidIdent

Synopsis: Check whether a string is a valid identifier name.

Declaration: `function IsValidIdent(const Ident: string) : 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: `./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.

```

37.14.165 LastDelimiter

Synopsis: Return the last occurrence of a set of delimiters in a string.

Declaration: `function LastDelimiter(const Delimiters: string;const S: string)
: Integer`

Visibility: default

Description: `LastDelimiter` returns the *last* occurrence of any character in the set `Delimiters` in the string `S`.

Listing: ./sysutex/ex88.pp

Program example88;

{ This program demonstrates the LastDelimiter function }

uses SysUtils;

begin

WriteLn(LastDelimiter('\.: ', 'c:\filename.ext'));
end.

37.14.166 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` ([1478](#)), `TrimLeft` ([1510](#)), `TrimRight` ([1510](#)), `Trim` ([1509](#))

Listing: ./sysutex/ex76.pp

Program Example76;

{ This program demonstrates the LeftStr function }

Uses sysutils;

Begin

WriteLn (LeftStr ('abcdefghijklmnopqrstuvwxy', 20));

```

WriteLn ( LeftStr ( 'abcdefghijklmnopqrstuvwxyz ' ,15));
WriteLn ( LeftStr ( 'abcdefghijklmnopqrstuvwxyz ' ,1));
WriteLn ( LeftStr ( 'abcdefghijklmnopqrstuvwxyz ' ,200));
End.

```

37.14.167 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.

37.14.168 LowerCase

Synopsis: Return a lowercase version of a string.

Declaration: `function LowerCase(const s: string) : string; Overload`
`function LowerCase(const V: variant) : string; 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.

Errors: None.

See also: `AnsiLowerCase` ([1392](#)), `UpperCase` ([1516](#)), `AnsiUpperCase` ([1400](#))

Listing: `./sysutex/ex77.pp`

Program `Example77;`

{ This program demonstrates the LowerCase function }

Uses `sysutils;`

Begin

`WriteLn (LowerCase('THIS WILL COME out all LoWeRcAsE !'));`
End.

37.14.169 MSecsToTimeStamp

Synopsis: Convert a number of milliseconds to a `TDateTime` value.

Declaration: `function MSecsToTimeStamp(MSecs: Comp) : TTimeStamp`

Visibility: default

Description: `MSecsTiTimeStamp` 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 \(1508\)](#), [DateTimeToTimeStamp \(1415\)](#)

Listing: ./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.

37.14.170 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 \(1402\)](#).

Errors: If not enough memory is present, an `EOutOfMemory` exception will be raised.

See also: [AssignStr \(1402\)](#), [DisposeStr \(1420\)](#)

37.14.171 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 \(1412\)](#), [Time \(1507\)](#)

Listing: ./sysutex/ex18.pp

```

Program Example18;

{ This program demonstrates the Now function }

Uses sysutils;

Begin
  WriteLn ( 'Now : ', DateTimeToStr(Now) );
End.

```

37.14.172 OutOfMemoryError

Synopsis: Raise an EOutOfMemory exception

Declaration: `procedure OutOfMemoryError`

Visibility: default

Description: `OutOfMemoryError` raises an `EOutOfMemory` ([1525](#)) exception, with an exception object that has been allocated on the heap at program startup. The program should never create an `EOutOfMemory` ([1525](#)) exception, but always call this routine.

See also: `EOutOfMemory` ([1525](#))

37.14.173 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` ([1393](#)), `AnsiExtractQuotedStr` ([1391](#))

Listing: ./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.

```

37.14.174 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` ([1525](#)) 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 ([1527](#)) will be raised.

See also: `EOSError` ([1525](#)), `GetLastError` ([1463](#)), Exception ([1527](#))

37.14.175 RemoveDir

Synopsis: Remove a directory from the filesystem.

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` ([1410](#)).

Errors: In case of error (e.g. the directory isn't empty) the function returns `False`. If successful, `True` is returned.

37.14.176 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 accross disks or partitions depends entirely on the OS. On unix-like OS-es, the rename function will fail when used accross partitions. On Windows, it will work.

Errors: On Error, False is returned.

See also: DeleteFile ([1418](#))

Listing: ./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.

37.14.177 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 ([1478](#))

37.14.178 ReplaceTime

Synopsis: Replace the time part

Declaration: procedure ReplaceTime (var dati: TDateTime; NewTime: TDateTime)

Visibility: default

Description: ReplaceTime replaces the time part in dati with NewTime. The date part remains untouched.

37.14.179 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). If Count is larger than the actual length of S only the real length will be used.

Errors: None.

See also: [LeftStr \(1473\)](#), [Trim \(1509\)](#), [TrimLeft \(1510\)](#), [TrimRight \(1510\)](#)

Listing: ./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.

37.14.180 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.

37.14.181 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` ([1388](#)) with arguments `S1` and `S2` returns 0, and returns `False` otherwise.

Errors: None.

See also: [AnsiCompareFileName \(1388\)](#)

37.14.182 SameText

Synopsis: Checks whether 2 strings are the same (case insensitive)

Declaration: `function SameText(const s1: string;const s2: string) : Boolean`

Visibility: default

Description: `SameText` calls `CompareText` ([1408](#)) 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 \(1408\)](#), [AnsiSameText \(1394\)](#), [AnsiSameStr \(1394\)](#)

37.14.183 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` ([1460](#))

37.14.184 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` ([1426](#)), `ExtractFilePath` ([1429](#)), `ExtractFileDir` ([1428](#))

Listing: `./sysutex/ex47.pp`

Program `Example47;`

{ This program demonstrates the SetDirSeparators function }

Uses `sysutils;`

Begin

`WriteLn (SetDirSeparators ('/pp\bin\win32\ppc386'));`

End.

37.14.185 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` ([1423](#)) 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` ([1385](#)) is executed.

Errors: If, for a GUI application, `OnShowException` ([1385](#)) 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` ([1423](#)), `ExceptAddr` ([1422](#)), `ExceptionErrorMessage` ([1423](#))

37.14.186 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.

37.14.187 SScanf

Synopsis: Scan a string for substrings and return the substrings

Declaration: `function SScanf(const s: string; const fmt: string;
const Pointers: Array of Pointer) : Integer`

Visibility: default

Description: `SScanF` scans the string `S` for the elements specified in `Fmt`, and returns the elements in the pointers in `Pointers`. The `Fmt` can contain placeholders of the form `%X` where `X` can be one of the following characters:

dPlaceholder for a decimal number.

fPlaceholder for a floating point number (an extended)

sPlaceholder for a string of arbitrary length.

cPlaceholder 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:

dA pointer to an integer.

fA pointer to an extended.

sA pointer to an ansistring.

cA pointer to a single character.

Errors: No error checking is performed on the type of the memory location.

See also: `Format` ([1449](#))

37.14.188 StrAlloc

Synopsis: Allocate a null-terminated string on the heap.

Declaration: `function StrAlloc(Size: Cardinal) : 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.

Additionally, `StrAlloc` allocates 4 extra bytes to store the size of the allocated memory. Therefore this function is NOT compatible with the `StrAlloc` (1075) function of the `Strings` unit.

For an example, see `StrBufSize` (1482).

Errors: None.

See also: `StrBufSize` (1482), `StrDispose` (1485), `StrAlloc` (1075)

37.14.189 StrBufSize

Synopsis: Return the size of a null-terminated string allocated on the heap.

Declaration: `function StrBufSize(Str: PChar) : Cardinal`

Visibility: default

Description: `StrBufSize` returns the memory allocated for `Str`. This function ONLY gives the correct result if `Str` was allocated using `StrAlloc` (1482).

Errors: If no more memory is available, a runtime error occurs.

See also: `StrAlloc` (1482), `StrDispose` (1485)

Listing: `./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.

37.14.190 StrByteType

Synopsis: Return the type of byte in a null-terminated string for a multi-byte character set

Declaration: `function StrByteType(Str: PChar; Index: Cardinal) : 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` ([1380](#)), `ByteType` ([1404](#))

37.14.191 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` ([1488](#))

Listing: `./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.

37.14.192 StrCharLength

Synopsis: Return the length of a null-terminated string in characters.

Declaration: `function StrCharLength(const Str: PChar) : Integer`

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.

37.14.193 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 Longint when S1<S2.
- 0 when S1=S2.
- A positive Longint when S1>S2.

For an example, see StrLComp ([1489](#)).

Errors: None.

See also: StrLComp ([1489](#)), StrIComp ([1487](#)), StrLComp ([1491](#))

37.14.194 StrCopy

Synopsis: Copy a null-terminated string

Declaration: `function strcpy(dest: PChar;source: PChar) : PChar; 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 ([1494](#)), StrLCopy ([1489](#)), StrECopy ([1485](#))

Listing: ./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.

37.14.195 StrDispose

Synopsis: Dispose of a null-terminated string on the heap.

Declaration: `procedure StrDispose(Str: PChar)`

Visibility: default

Description: `StrDispose` frees any memory allocated for `Str`. This function will only function correctly if `Str` has been allocated using `StrAlloc` (1482) from the `SysUtils` unit.

For an example, see `StrBufSize` (1482).

Errors: If an invalid pointer is passed, or a pointer not allocated with `StrAlloc`, an error may occur.

See also: `StrBufSize` (1482), `StrAlloc` (1482), `StrDispose` (1485)

37.14.196 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` (1489), `StrCopy` (1484)

Listing: `./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.

37.14.197 strend

Synopsis: Return a pointer to the end of a null-terminated string

Declaration: `function strend(p: PChar) : PChar`

37.14.199 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 Longint when S1<S2.
- 0 when S1=S2.
- A positive Longint when S1>S2.

Errors: None.

See also: StrLComp ([1489](#)), StrComp ([1484](#)), StrLComp ([1491](#))

Listing: ./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.

37.14.200 StringReplace

Synopsis: Replace occurrences of one substring with another in a string.

Declaration: `function StringReplace(const S: string;const OldPattern: string;
 const NewPattern: string;Flags: TReplaceFlags)
 : string`

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 runed with Flags, which is of type TReplaceFlags ([1380](#)). Standard behaviour is to replace only the first occurrence of OldPattern, and to search case sensitively.

Errors: None.

See also: TReplaceFlags ([1380](#))

37.14.201 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` ([1521](#)) exception is raised.

See also: `Supports` ([1504](#)), `#rtl.system.TGUID` ([1166](#)), `GUIDToString` ([1466](#)), `IsEqualGuid` ([1471](#))

37.14.202 strlcat

Synopsis: Concatenate 2 null-terminated strings, with length boundary.

Declaration: `function strlcat(dest: PChar;source: PChar;l: SizeInt) : PChar`

Visibility: default

Description: Adds `MaxLen` characters from `Source` to `Dest`, and adds a terminating null-character. Returns `Dest`.

Errors: None.

See also: `StrCat` ([1483](#))

Listing: `./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.

37.14.203 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 Longint when S1<S2.
- 0 when S1=S2.
- A positive Longint when S1>S2.

Errors: None.

See also: StrComp ([1484](#)), StrIComp ([1487](#)), StrLComp ([1491](#))

Listing: ./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.

37.14.204 StrLCopy

Synopsis: Copy a null-terminated string, limited in length.

Declaration: `function strlcopy(dest: PChar;source: PChar;maxlen: SizeInt) : PChar`
 ; 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 ([1484](#)), StrECopy ([1485](#))

Listing: ./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.

37.14.205 StrLen

Synopsis: Length of a null-terminated string.

Declaration: function strlen(p: PChar) : 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 ([1493](#))

Listing: ./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.

37.14.206 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` (1449) 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` (1449).

See also: `StrFmt` (1486), `FmtStr` (1448), `Format` (1449), `FormatBuf` (1456)

Listing: `./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 fomatting see %s.',['Format']));

End.

37.14.207 `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 `Longint` when `S1<S2`.
- 0 when `S1=S2`.
- A positive `Longint` when `S1>S2`.

For an example, see `StrIComp` (1487)

Errors: None.

See also: `StrLComp` (1489), `StrComp` (1484), `StrIComp` (1487)

37.14.208 `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 \(1504\)](#)

Listing: ./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

```
WriteIn ( 'Uppercase : ',StrUpper(P2));
```

```
StrLower (P1);
```

```
WriteLn ( 'Lowercase : ',P1 );
```

end .

37.14.209 strmove

Synopsis: Move a null-terminated string to new location.

```
Declaration: function strmove(dest: PChar;source: PChar;l: SizeInt) : PChar
                ; Overload
```

Visibility: default

Description: Copies `MaxLen` characters from `Source` to `Dest`. No terminating null-character is copied. Returns `Dest`

Errors: None.

See also: [StrLCopy \(1489\)](#), [StrCopy \(1484\)](#)

Listing: `./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 }
```

```
WriteIn ( 'P2 = ',P2);
```

```
StrDispose (P2);
```

end.

37.14.210 strnew

Synopsis: Allocate room for new null-terminated string.

Declaration: `function strnew(p: PChar) : PChar; 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` ([1484](#)), `StrDispose` ([1485](#))

Listing: `./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.

37.14.211 StrNextChar

Synopsis: Returns a pointer to the location of the next empty character in a null-terminated string

Declaration: `function StrNextChar(const Str: PChar) : PChar`

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.

37.14.212 StrPas

Synopsis: Convert a null-terminated string to an ansistring.

Declaration: `function StrPas(Str: PChar) : string; 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` ([1494](#)), `StrPLCopy` ([1494](#))

37.14.213 StrPCopy

Synopsis: Copy an ansistring to a null-terminated string.

Declaration: `function StrPCopy(Dest: PChar; const Source: string) : PChar; 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` ([1494](#)), `StrPas` ([1493](#))

37.14.214 StrPLCopy

Synopsis: Copy a limited number of characters from an ansistring to a null-terminated string.

Declaration: `function StrPLCopy(Dest: PChar; const Source: string; MaxLen: SizeUInt) : PChar; 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` ([1494](#))

37.14.215 strpos

Synopsis: Find position of one null-terminated substring in another.

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` ([1495](#)), `StrRScan` ([1495](#))

Listing: `./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.

37.14.216 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 ([1495](#)).

Errors: None.

See also: StrScan ([1495](#)), StrPos ([1494](#))

37.14.217 strscan

Synopsis: Find first occurrence of a character in a null-terminated string.

Declaration: `function strscan(p: PChar; c: Char) : PChar; 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 ([1495](#)), StrPos ([1494](#))

Listing: ./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.

37.14.218 StrToBool

Synopsis: Convert a string to a boolean value

Declaration: `function StrToBool(const S: string) : 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 ([1521](#)) exception is raised.

Errors: On error, an `EConvertError` ([1521](#)) exception is raised.

See also: `BoolToStr` ([1403](#))

37.14.219 `StrToBoolDef`

Synopsis: Convert string to boolean value, returning default in case of error

Declaration: `function StrToBoolDef(const S: string; Default: Boolean) : 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` ([1495](#)), `TryStrToBool` ([1512](#))

37.14.220 `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` ([1521](#)) exception is raised.

Errors: On error, an `EConvertError` ([1521](#)) exception is raised.

See also: `CurrToStr` ([1411](#)), `StrToCurrDef` ([1496](#))

37.14.221 `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` ([1411](#)), `StrToCurr` ([1496](#))

37.14.222 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: Char) : TDateTime
function StrToDate(const S: AnsiString;separator: Char) : TDateTime
function StrToDate(const S: string;FormatSettings: TFormatSettings)
    : TDateTime
function StrToDate(const S: ShortString;const useformat: string;
    separator: Char) : TDateTime
function StrToDate(const S: AnsiString;const useformat: string;
    separator: Char) : TDateTime
function StrToDate(const S: PChar;Len: Integer;const useformat: string;
    separator: Char) : 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` exception is raised.

See also: `StrToTime` ([1503](#)), `DateToStr` ([1415](#)), `TimeToStr` ([1509](#))

Listing: ./sysutex/ex19.pp

Program Example19;

{ This program demonstrates the StrToDate function }

Uses sysutils;

Procedure TestStr (S : **String**);

begin

WriteIn (S, ' : ', **DateToStr**(**StrToDate**(S)));
end;

Begin

WriteIn ('ShortDateFormat ', ShortDateFormat);
 TestStr(**DateTimeToStr**(**Date**));
 TestStr('05'+DateSeparator+'05'+DateSeparator+'1999');
 TestStr('5'+DateSeparator+'5');
 TestStr('5');
End.

37.14.223 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: Char) : TDateTime
function StrToDateDef(const S: AnsiString;const Defvalue: TDateTime)
    : TDateTime
function StrToDateDef(const S: AnsiString;const Defvalue: TDateTime;
    separator: Char) : 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 ([1497](#)), TryStrToDate ([1513](#)), StrToTimeDef ([1504](#))

37.14.224 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 ([1497](#)) 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 exception is raised.

See also: StrToDate ([1497](#)), StrToTime ([1503](#)), DateTimeToStr ([1413](#))

Listing: ./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

```

WriteIn ( 'ShortDateFormat ', ShortDateFormat );
TestStr( DateTimeToStr(Now) );
TestStr( '05-05-1999 15:50' );
TestStr( '5-5 13:30' );
TestStr( '5 1:30PM' );
End.

```

37.14.225 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

```

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` ([1504](#)), `StrToDateDef` ([1497](#)), `TryStrToDateTime` ([1513](#)), `StrToDateTime` ([1498](#))

37.14.226 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` ([1506](#)), `FloatToStr` ([1444](#)), `FormatFloat` ([1457](#)), `StrToInt` ([1500](#))

Listing: `./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;

end;

Begin
    DecimalSeparator:=',';
    Testit;
    DecimalSeparator:= '.';
    Testit;

End.

```

37.14.227 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.

37.14.228 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 an integer. If the string contains invalid characters or has an invalid format, then an `EConvertError` 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 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.

Errors: In case of error, an `EConvertError` is raised.

See also: `IntToStr` ([1470](#)), `StrToIntDef` ([1502](#))

Listing: `./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.

37.14.229 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` ([1521](#)) exception is raised.

See also: `TryStrToInt64` ([1515](#)), `StrToInt64Def` ([1502](#)), `StrToInt` ([1500](#)), `TryStrToInt` ([1514](#)), `StrToIntDef` ([1502](#))

37.14.230 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` ([1501](#)), `TryStrToInt64` ([1515](#)), `StrToInt` ([1500](#)), `TryStrToInt` ([1514](#)), `StrToIntDef` ([1502](#))

37.14.231 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` ([1470](#)), `StrToInt` ([1500](#))

Listing: `./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.

37.14.232 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 (1521)` exception is raised.

See also: [TryStrToQWord \(1515\)](#), [StrToQWordDef \(1503\)](#), [StrToInt64 \(1501\)](#), [StrToInt \(1500\)](#)

37.14.233 StrToQWordDef

Synopsis: Try to convert a string to a QWord, returning a default value in case of failure.

Declaration: `function StrToQWordDef(const S: string; Default: QWord) : QWord`

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 \(1503\)](#), [TryStrToQWord \(1515\)](#), [StrToInt64Def \(1502\)](#), [StrToIntDef \(1502\)](#)

37.14.234 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: Char) : TDateTime
              function StrToTime(const S: AnsiString;separator: Char) : TDateTime
              function StrToTime(const S: string;FormatSettings: TFormatSettings)
                  : TDateTime
              function StrToTime(const S: PChar;Len: Integer;separator: Char)
                  : 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` exception is raised.

See also: [StrToDate \(1497\)](#), [StrToDateTime \(1498\)](#), [TimeToStr \(1509\)](#)

Listing: ./sysutex/ex21.pp

Program Example21 ;

```
{ This program demonstrates the StrToTime function }
```

Uses sysutils;

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 ([1488](#))

37.14.238 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 ([1525](#))

37.14.239 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 ([1414](#))

Listing: ./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.

37.14.240 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 stroing representation of a floating point value (either in decimal or scientific notation). If the buffer contains a decimal value, then the decimal separator character can either be a '.' or the value of the DecimalSeparator variable.

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 ([1499](#)), FloatToStr ([1444](#)), FormatFloat ([1457](#))

Listing: ./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.
```

37.14.241 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 ([1475](#)), Date ([1412](#))

Listing: ./sysutex/ex23.pp

Program Example23;

```
{ This program demonstrates the Time function }
```

```
Uses sysutils;
```

```
Begin
```

```
  WriteLn ( 'The time is : ', TimeToStr(Time));
```

```
End.
```

37.14.242 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` (1415).

Errors: None.

See also: `DateTimeToTimeStamp` (1415), `TimeStampToMSecs` (1508)

Listing: `./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.

37.14.243 TimeStampToMSecs

Synopsis: Converts a timestamp to a number of milliseconds.

Declaration: `function TimeStampToMSecs(const TimeStamp: TTimeStamp) : comp`

Visibility: default

Description: `TimeStampToMSecs` converts `TimeStamp` to the number of seconds since 1/1/0001.

Use `TTimeStamp` variables if you need to keep very precise track of time.

For an example, see `MSecsToTimeStamp` (1474).

Errors: None.

See also: `MSecsToTimeStamp` (1474), `TimeStampToDateTime` (1508)

37.14.244 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.

Errors: None.

Listing: `./sysutex/ex25.pp`

Program `Example25;`

{ This program demonstrates the TimeToStr function }

Uses `sysutils;`

Begin

`WriteLn ('The current time is : ', TimeToStr(Time));`

End.

37.14.245 Trim

Synopsis: Trim whitespace from the ends of a string.

Declaration: `function Trim(const S: string) : string`
`function Trim(const S: widestring) : widestring`

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` ([1510](#)), `TrimRight` ([1510](#))

Listing: `./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.

```

37.14.246 TrimLeft

Synopsis: Trim whitespace from the beginning of a string.

Declaration: `function TrimLeft(const S: string) : string`
`function TrimLeft(const S: wstring) : wstring`

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` ([1509](#)), `TrimRight` ([1510](#))

Listing: `./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.

```

37.14.247 TrimRight

Synopsis: Trim whitespace from the end of a string.

Declaration: `function TrimRight(const S: string) : string`
`function TrimRight(const S: wstring) : wstring`

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` ([1509](#)), `TrimLeft` ([1510](#))

Listing: `./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.

37.14.248 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` ([1421](#)), `DecodeDateFully` ([1417](#)), `DecodeDate` ([1416](#)), `TryEncodeTime` ([1511](#))

37.14.249 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` ([1422](#)), `DecodeTime` ([1417](#)), `TryEncodeDate` ([1511](#))

37.14.250 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` ([1371](#)) and `MaxCurrency` ([1371](#))). If not, `False` is returned.

Errors: If `Value` is out of range, `False` is returned.

See also: `FloatToCurr` ([1443](#)), `MinCurrency` ([1371](#)), `MaxCurrency` ([1371](#))

37.14.251 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` ([1488](#))

37.14.252 TryStrToBool

Synopsis: Try to convert a string to a boolean value

Declaration: `function TryStrToBool(const S: string; out Value: Boolean) : 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 indetermined.

Valid boolean string constants are in the `FalseBoolStrs` ([1383](#)) (for `False` values) and `TrueBoolStrs` ([1386](#)) (for `True` values) variables.

See also: `StrToBool` ([1495](#)), `StrToBoolDef` ([1496](#))

37.14.253 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` ([1496](#)), which raises an exception when the conversion fails.

The function takes into account locale information.

See also: `StrToCurr` ([1496](#)), `TextToFloat` ([1506](#))

37.14.254 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: Char) : Boolean`
`function TryStrToDate(const S: AnsiString;out Value: TDateTime;`
`separator: Char) : Boolean`
`function TryStrToDate(const S: ShortString;out Value: TDateTime;`
`const useformat: string;separator: Char) : Boolean`
`function TryStrToDate(const S: AnsiString;out Value: TDateTime;`
`const useformat: string;separator: Char) : 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` ([1497](#)), `StrToTime` ([1503](#)), `TryStrToTime` ([1515](#)), `TryStrToDateTime` ([1513](#)), `DateToStr` ([1415](#)), `TimeToStr` ([1509](#))

37.14.255 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` (1513), `TryStrToTime` (1515), `StrToDateTime` (1498), `StrToTime` (1503), `DateToStr` (1415), `TimeToStr` (1509)

37.14.256 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
```

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 succesful, 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` (1499)

37.14.257 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 succesful. 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` (1500), `TryStrToInt64` (1515), `StrToIntDef` (1502), `StrToInt64` (1501), `StrToInt64Def` (1502)

37.14.258 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` ([1501](#)), `StrToInt64Def` ([1502](#)), `StrToInt` ([1500](#)), `TryStrToInt` ([1514](#)), `StrToIntDef` ([1502](#))

37.14.259 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` ([1503](#)), `StrToQWordDef` ([1503](#)), `TryStrToInt64` ([1515](#)), `TryStrToInt` ([1514](#))

37.14.260 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: Char) : Boolean
function TryStrToTime(const S: AnsiString; out Value: TDateTime;
                      separator: Char) : 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` ([1513](#)), `TryStrToDateTime` ([1513](#)), `StrToDate` ([1497](#)), `StrToTime` ([1503](#)), `DateToStr` ([1415](#)), `TimeToStr` ([1509](#))

37.14.261 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.

37.14.262 UpperCase

Synopsis: Return an uppercase version of a string.

Declaration: `function UpperCase(const s: string) : string; 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 compatiibility only.

Errors: None.

See also: [AnsiLowerCase \(1392\)](#), [LowerCase \(1474\)](#), [AnsiUpperCase \(1400\)](#)

Listing: `./sysutex/ex87.pp`

Program `Example87;`

{ This program demonstrates the UpperCase function }

Uses `sysutils;`

Begin

`WriteLn (UpperCase('this will come OUT ALL uPpErCaSe !'));`

End.

37.14.263 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 (1385)` event must be set, and an appropriate return value must be returned. The Vendor name is used in `GetAppConfigDir (1459)` and `GetAppConfigFile (1460)` to determine the configuration directory.

Errors: If `OnGetVendorName (1385)` is not set, an empty string is returned.

See also: [OnGetVendorName \(1385\)](#), [GetAppConfigDir \(1459\)](#), [GetAppConfigFile \(1460\)](#)

37.14.264 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` (1517), the comparison is case sensitive.

Errors: None.

See also: `WideCompareText` (1517), `WideSameStr` (1519), `WideSameText` (1519)

37.14.265 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` (1517), the comparison is case insensitive.

Errors: None.

See also: `WideCompareStr` (1517), `WideSameStr` (1519), `WideSameText` (1519)

37.14.266 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` (1518), `WideFormatBuf` (1518), `Format` (1449)

37.14.267 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` (1449) but accepts as a formatting string a `WString`. The resulting string is also a `WString`.

For more information about the used formatting characters, see the `Format` (1449) string.

See also: `Format` (1449)

37.14.268 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` (1518) 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` (1517), `WideFormat` (1518), `Format` (1449), `FormatBuf` (1456)

37.14.269 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` (1519)

37.14.270 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` (1517) returns 0 (zero), i.e. when `S1` and `S2` are the same string (taking into account case).

See also: `WideSameText` (1519), `WideCompareStr` (1517), `WideCompareText` (1517), `AnsiSameStr` (1394)

37.14.271 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` (1517) returns 0 (zero), i.e. when `S1` and `S2` are the same string (taking into account case).

See also: `WideSameStr` (1519), `WideCompareStr` (1517), `WideCompareText` (1517), `AnsiSameText` (1394)

37.14.272 WideUpperCase

Synopsis: Change a widestring to all-uppercase.

Declaration: `function WideUpperCase(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` (1518)

37.14.273 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).

See also: `StringReplace` (1487)

37.15 EAbort

37.15.1 Description

`EAbort` is raised by the `Abort` ([1387](#)) 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` ([1387](#))

37.16 EAbstractError

37.16.1 Description

`EAbstractError` is raised when an abstract error occurs, i.e. when an unimplemented abstract method is called.

37.17 EAccessViolation

37.17.1 Description

`EAccessViolation` is raised when the OS reports an Access Violation, i.e. when invalid memory is accessed.

See also: `EObjectCheck` ([1524](#))

37.18 EArgumentException

37.18.1 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` ([1520](#))

37.19 EArgumentOutOfRangeException

37.19.1 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` ([1520](#))

37.20 EAssertionFailed

37.20.1 Description

EAssertionFailed is raised when an application that is compiled with assertions, encounters an invalid assertion.

37.21 EBusError

37.21.1 Description

EBusError is raised in case of a bus error.

37.22 EControlC

37.22.1 Description

EControlC is raised when the user has pressed CTRL-C in a console application.

37.23 EConvertError

37.23.1 Description

EConvertError is raised by the various conversion routines in the SysUtils unit. The message will contain more specific error information.

37.24 EDivByZero

37.24.1 Description

EDivByZero is used when the operating system or CPU signals a division by zero error.

37.25 EExternal

37.25.1 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 descendents of EExternal.

See also: EInterror ([1522](#)), EExternal ([1521](#)), EMathError ([1524](#)), EExternalException ([1521](#)), EAccessViolation ([1520](#)), EPrivilege ([1525](#)), EStackOverflow ([1526](#)), EControlC ([1521](#))

37.26 EExternalException

37.26.1 Description

EExternalException is raised when an external routine raises an exception.

See also: [EExternal \(1521\)](#)

37.27 EFormatError

37.27.1 Description

`EFormatError` is raised in case of an error in one of the various `Format` ([1449](#)) functions.

See also: `Format` ([1449](#))

37.28 EHeapMemoryError

37.28.1 Description

`EHeapMemoryError` is raised when an error occurs in heap (dynamically allocated) memory.

See also: `EHeapException` ([1375](#)), `EoutOfMemory` ([1525](#)), `EInvalidPointer` ([1523](#))

37.28.2 Method overview

Page	Property	Description
1522	<code>FreeInstance</code>	Free the exception instance

37.28.3 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.

37.29 EInOutError

37.29.1 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` (??)

37.30 EIntError

37.30.1 Description

`EIntError` is used when the operating system or CPU signals an integer operation error, e.g., an overflow.

37.31 EIntfCastError

37.31.1 Description

`EIntfCastError` is raised when an invalid interface cast is encountered.

See also: `EInvalidCast` ([1523](#))

37.32 EIntOverflow

37.32.1 Description

`EIntOverflow` is used when the operating system or CPU signals a integer overflow error.

See also: `EIntError` ([1522](#)), `EDivByZero` ([1521](#)), `ERangeError` ([1526](#))

37.33 EInvalidCast

37.33.1 Description

`EInvalidCast` is raised when an invalid typecast error (using the `as` operator) is encountered.

See also: `EIntfCastError` ([1523](#))

37.34 EInvalidContainer

37.34.1 Description

`EInvalidContainer` is not yet used by Free Pascal, and is provided for Delphi compatibility only.

37.35 EInvalidInsert

37.35.1 Description

`EInvalidInsert` is not yet used by Free Pascal, and is provided for Delphi compatibility only.

37.36 EInvalidOp

37.36.1 Description

`EInvalidOp` is raised when an invalid operation is encountered.

37.37 EInvalidPointer

37.37.1 Description

`EInvalidPointer` is raised when an invalid heap pointer is used.

See also: [EHeapException \(1375\)](#), [EHeapMemoryError \(1522\)](#), [EOutOfMemory \(1525\)](#)

37.38 EMathError

37.38.1 Description

`EMathError` is used when the operating system or CPU signals a floating point overflow error.

See also: [EIntError \(1522\)](#), [EIntOverflow \(1523\)](#), [EDivByZero \(1521\)](#), [ERangeError \(1526\)](#)

37.39 ENoConstructException

37.39.1 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.

37.40 ENoThreadSupport

37.40.1 Description

`ENoThreadSupport` is raised when some thread routines are invoked, and thread support was not enabled when the program was compiled.

37.41 ENotImplemented

37.41.1 Description

`ENotImplemented` can be used to raise an exception when a particular call had been defined, but was not implemented.

37.42 ENoWideStringSupport

37.42.1 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.

37.43 EObjectCheck

37.43.1 Description

`EObjectCheck` is raised when the `-CR` (check object references) command-line option or `{ $OBJECTCHECKS ON }` directive is in effect and a `Nil` reference to an object or class was encountered.

See also: [EAccessViolation \(1520\)](#)

37.44 EOSError

37.44.1 Description

`EOSError` is raised when some Operating System call fails. The `ErrorCode` (??) property contains the operating system error code.

See also: `EOSError.ErrorCode` (??)

37.45 EOutOfMemory

37.45.1 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` ([1375](#)), `EHeapMemoryError` ([1522](#)), `EInvalidPointer` ([1523](#))

37.46 EOverflow

37.46.1 Description

`EOverflow` occurs when a float operation overflows. (i.e. result is too big to represent).

See also: `EIntError` ([1522](#)), `EIntOverflow` ([1523](#)), `EDivByZero` ([1521](#)), `ERangeError` ([1526](#)), `EUnderFlow` ([1526](#))

37.47 EPackageError

37.47.1 Description

`EPackageError` is not yet used by Free Pascal, and is provided for Delphi compatibility only.

37.48 EPrivilege

37.48.1 Description

`EPrivilege` is raised when the OS reports that an invalid instruction was executed.

37.49 EPropReadOnly

37.49.1 Description

`EPropReadOnly` is raised when an attempt is made to write to a read-only property.

37.50 EPropWriteOnly

37.50.1 Description

EPropWriteOnly is raised when an attempt is made to read from a write-only property.

See also: EPropReadOnly ([1525](#))

37.51 ERangeError

37.51.1 Description

ERangeError is raised by the Free Pascal runtime library if range checking is on, and a range check error occurs.

See also: EIntError ([1522](#)), EDivByZero ([1521](#)), EIntOverflow ([1523](#))

37.52 ESafecallException

37.52.1 Description

ESafecallException is not yet used by Free Pascal, and is provided for Delphi compatibility only.

37.53 EStackOverflow

37.53.1 Description

EStackOverflow occurs when the stack has grown too big (e.g. by infinite recursion).

37.54 EUnderflow

37.54.1 Description

EOverflow occurs when a float operation underflows (i.e. result is too small to represent).

See also: EIntError ([1522](#)), EIntOverflow ([1523](#)), EDivByZero ([1521](#)), ERangeError ([1526](#)), EOverflow ([1525](#))

37.55 EVariantError

37.55.1 Description

EVariantError is raised by the internal variant routines.

37.55.2 Method overview

Page	Property	Description
1527	CreateCode	Create an instance of EVariantError with a particular error code.

37.55.3 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` (??)

37.56 Exception

37.56.1 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` (1423), `ExceptAddr` (1422), `ExceptionErrorMessage` (1423), `ShowException` (1480), `Abort` (1387)

37.56.2 Method overview

Page	Property	Description
1527	<code>Create</code>	Constructs a new exception object with a given message.
1528	<code>CreateFmt</code>	Constructs a new exception object and formats a new message.
1529	<code>CreateFmtHelp</code>	Constructs a new exception object and sets the help context and formats the message
1528	<code>CreateHelp</code>	Constructs a new exception object and sets the help context.
1528	<code>CreateRes</code>	Constructs a new exception object and gets the message from a resource.
1528	<code>CreateResFmt</code>	Constructs a new exception object and formats the message from a resource.
1529	<code>CreateResFmtHelp</code>	Constructs a new exception object and sets the help context and formats the message from a resource
1529	<code>CreateResHelp</code>	Constructs a new exception object and sets the help context and gets the message from a resource
1529	<code>ToString</code>	Nicely formatted version of the exception message

37.56.3 Property overview

Page	Property	Access	Description
1530	<code>HelpContext</code>	<code>rw</code>	Help context associated with the exception.
1530	<code>Message</code>	<code>rw</code>	Message associated with the exception.

37.56.4 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` ([1528](#)), `Exception.Message` ([1530](#))

37.56.5 `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` ([1527](#)), `Exception.Message` ([1530](#)), `Format` ([1449](#))

37.56.6 `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` ([1527](#)), `Exception.CreateFmt` ([1528](#)), `Exception.CreateResFmt` ([1528](#)), `Exception.Message` ([1530](#))

37.56.7 `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` ([1528](#)), 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` ([1527](#)), `Exception.CreateFmt` ([1528](#)), `Exception.CreateRes` ([1528](#)), `Exception.Message` ([1530](#))

37.56.8 `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` ([1527](#)) constructor, but additionally stores `AHelpContext` in the `HelpContext` ([1530](#)) property.

See also: `Exception.Create` ([1527](#))

37.56.9 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` (1528) constructor, but additionally stores `AHelpContext` in the `HelpContext` (1530) property.

See also: `Exception.CreateFmt` (1528)

37.56.10 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` (1528) constructor, but additionally stores `AHelpContext` in the `HelpContext` (1530) property.

See also: `Exception.CreateRes` (1528)

37.56.11 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` (1528) constructor, but additionally stores `AHelpContext` in the `HelpContext` (1530) property.

See also: `Exception.CreateResFmt` (1528)

37.56.12 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` (1530).

See also: `Exception.Message` (1530)

37.56.13 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` ([1528](#)), `Exception.Message` ([1530](#))

37.56.14 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` ([1480](#)) routine, and should be set in the constructor when the exception is raised.

See also: `Exception.Create` ([1527](#)), `Exception.HelpContext` ([1530](#))

37.57 EZeroDivide

37.57.1 Description

`EZeroDivide` occurs when a float division by zero occurs.

See also: `EIntError` ([1522](#)), `EIntOverflow` ([1523](#)), `EDivByZero` ([1521](#)), `ERangeError` ([1526](#))

37.58 IReadWriteSync

37.58.1 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.

37.59 TMultiReadExclusiveWriteSynchronizer

37.59.1 Description

`TMultiReadExclusiveWriteSynchronizer` is a default implementation of the `IReadWriteSync` ([1530](#)) 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` ([1530](#))

37.59.2 Interfaces overview

Page	Property	Description
1530	IReadWriteSync	Read/Write synchronizer

37.60 TSimpleRWSync

37.60.1 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` ([1530](#))

37.60.2 Interfaces overview

Page	Property	Description
1530	IReadWriteSync	Read/Write synchronizer

Chapter 38

Reference for unit 'types'

38.1 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.

38.2 Constants, types and variables

38.2.1 Constants

`E_FAIL = ($80004005)`

Defined for Delphi compatibility, this should not be used.

`E_INVALIDARG = ($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 = (\$800300FA)

Defined for Delphi compatibility, this should not be used.

STG_E_ACCESSDENIED = (\$80030005)

Defined for Delphi compatibility, this should not be used.

STG_E_BADBASEADDRESS = (\$80030110)

Defined for Delphi compatibility, this should not be used.

STG_E_CANTSAVE = (\$80030103)

Defined for Delphi compatibility, this should not be used.

STG_E_DISKISWRITEPROTECTED = (\$80030013)

Defined for Delphi compatibility, this should not be used.

STG_E_DOCFILECORRUPT = (\$80030109)

Defined for Delphi compatibility, this should not be used.

STG_E_EXTANTMARSHALLINGS = (\$80030108)

Defined for Delphi compatibility, this should not be used.

STG_E_FILEALREADYEXISTS = (\$80030050)

Defined for Delphi compatibility, this should not be used.

STG_E_FILENOTFOUND = (\$80030002)

Defined for Delphi compatibility, this should not be used.

STG_E_INCOMPLETE = (\$80030201)

Defined for Delphi compatibility, this should not be used.

STG_E_INSUFFICIENTMEMORY = (\$80030008)

Defined for Delphi compatibility, this should not be used.

STG_E_INUSE = (\$80030100)

Defined for Delphi compatibility, this should not be used.

STG_E_INVALIDFLAG = (\$800300FF)

Defined for Delphi compatibility, this should not be used.

STG_E_INVALIDFUNCTION = (\$80030001)

Defined for Delphi compatibility, this should not be used.

STG_E_INVALIDHANDLE = (\$80030006)

Defined for Delphi compatibility, this should not be used.

STG_E_INVALIDHEADER = (\$800300FB)

Defined for Delphi compatibility, this should not be used.

STG_E_INVALIDNAME = (\$800300FC)

Defined for Delphi compatibility, this should not be used.

STG_E_INVALIDPARAMETER = (\$80030057)

Defined for Delphi compatibility, this should not be used.

STG_E_INVALIDPOINTER = (\$80030009)

Defined for Delphi compatibility, this should not be used.

STG_E_LOCKVIOLATION = (\$80030021)

Defined for Delphi compatibility, this should not be used.

STG_E_MEDIUMFULL = (\$80030070)

Defined for Delphi compatibility, this should not be used.

STG_E_NOMOREFILES = (\$80030012)

Defined for Delphi compatibility, this should not be used.

STG_E_NOTCURRENT = (\$80030101)

Defined for Delphi compatibility, this should not be used.

STG_E_OLDDLL = (\$80030105)

Defined for Delphi compatibility, this should not be used.

STG_E_OLDFORMAT = (\$80030104)

Defined for Delphi compatibility, this should not be used.

STG_E_PATHNOTFOUND = (\$80030003)

Defined for Delphi compatibility, this should not be used.

STG_E_PROPSETMISMATCHED = (\$800300F0)

Defined for Delphi compatibility, this should not be used.

STG_E_READFAULT = (\$8003001E)

Defined for Delphi compatibility, this should not be used.

STG_E_REVERTED = (\$80030102)

Defined for Delphi compatibility, this should not be used.

STG_E_SEEKERROR = (\$80030019)

Defined for Delphi compatibility, this should not be used.

STG_E_SHAREREQUIRED = (\$80030106)

Defined for Delphi compatibility, this should not be used.

STG_E_SHAREVIOLATION = (\$80030020)

Defined for Delphi compatibility, this should not be used.

STG_E_TERMINATED = (\$80030202)

Defined for Delphi compatibility, this should not be used.

STG_E_TOOMANYOPENFILES = (\$80030004)

Defined for Delphi compatibility, this should not be used.

STG_E_UNIMPLEMENTEDFUNCTION = (\$800300FE)

Defined for Delphi compatibility, this should not be used.

STG_E_UNKNOWN = (\$800300FD)

Defined for Delphi compatibility, this should not be used.

STG_E_WRITEFAULT = (\$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.

38.2.2 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`

PWord 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 (1541) type.

PPoleStr = ^PoleStr

PPoleStr is a typed pointer to a PoleStr variable.

PRect = ^TRect

PRect is a typed pointer to the TRect (1541) type.

PSize = ^TSize

PSize is a typed pointer to the TSize (1542) type.

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 (1542) 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
```

tagPOINT is a simple alias for TPoint ([1541](#))

```
tagSIZE = TSize
```

tagSize is an alias for the TSize ([1542](#)) type.

```
tagSTATSTG = record
  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 ([1550](#)) call. It describes a storage medium (typically a file).

```
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.

TCLSID = TGUID

TCLSID is an alias for the #rtl.system.TGUID (1166) type.

TDoubleDynArray = Array of Double

TSoubleDynArray is a standard definition of a dynamical array of doubles. (regular floating point type)

TDuplicates = (dupIgnore, dupAccept, dupError)

Table 38.1: 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.

TFileTime = _FILETIME

Alias for the _FILETIME type

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 (1541)

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` (1540)

`TLongWordDynArray` = Array of `LongWord`

`TLongWordDynArray` is a standard definition of a dynamical array of (32-bit, unsigned) `LongWords`.

`TLeChar` = `WideChar`

`TLeChar` is an alias for the `WideChar` type, defined in the system unit.

```
TPoint = packed record
  X : LongInt;
  Y : LongInt;
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`.

`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`.

```
TRect = packed record
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.

`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)

```
TSize = packed record
  cx : LongInt;
  cy : 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.

```
TSmallIntDynArray = Array of SmallInt
```

TSmallIntDynArray is a standard definition of a dynamical array of (16-bit, unsigned) integers.

```
TSmallPoint = packed record
  x : SmallInt;
  y : SmallInt;
end
```

TSmallPoint defines a point in a 2-dimensional plane, just like TPoint ([1541](#)), but the coordinates have a smaller range: The coordinates are smallints (16-bit, signed) and they run from -MaxSmallInt to maxSmallInt.

```
TStatStg = tagSTATSTG
```

TStatStg is a record type describing a storage medium. It is used in the IStream.Stat ([1550](#)) function.

```
TStringDynArray = Array of AnsiString
```

TStringDynArray is a standard definition of a dynamical array of AnsiStrings.

```
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
  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.

38.3 Procedures and functions

38.3.1 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 (1545), PtInRect (1545), IntersectRect (1544), UnionRect (1546)

38.3.2 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 (1545), IntersectRect (1544), IsRectEmpty (1544), OffsetRect (1544), InflateRect (1544), Size (1545), IsRectEmpty (1544)

38.3.3 EqualRect

Synopsis: Check if two rectangles are equal.

Declaration: `function EqualRect (const r1: TRect; const r2: TRect) : 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 (1545), Bounds (1543), PtInRect (1545), IntersectRect (1544), UnionRect (1546), IsRectEmpty (1544), OffsetRect (1544), InflateRect (1544), Size (1545)

38.3.4 InflateRect

Synopsis: Increase the rectangle in size, keeping it centered

Declaration: `function InflateRect (var Rect: TRect; dx: Integer; dy: Integer) : 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` (1545), `IntersectRect` (1544), `IsRectEmpty` (1544), `OffsetRect` (1544), `CenterPoint` (1543), `Size` (1545), `IsRectEmpty` (1544)

38.3.5 IntersectRect

Synopsis: Return the intersection of 2 rectangles

Declaration: `function IntersectRect (var Rect: TRect; const R1: TRect; const R2: TRect) : 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` (1545), `UnionRect` (1546), `IsRectEmpty` (1544), `OffsetRect` (1544), `InflateRect` (1544), `Size` (1545)

38.3.6 IsRectEmpty

Synopsis: Check whether a rectangle is empty

Declaration: `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` (1545), `IntersectRect` (1544), `IsRectEmpty` (1544), `OffsetRect` (1544), `InflateRect` (1544), `Size` (1545)

38.3.7 OffsetRect

Synopsis: Offset the rectangle

Declaration: `function OffsetRect (var Rect: TRect; DX: Integer; DY: Integer) : 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` (1545), `IntersectRect` (1544), `IsRectEmpty` (1544), `OffsetRect` (1544), `InflateRect` (1544), `Size` (1545), `IsRectEmpty` (1544)

38.3.8 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` (1545), `PtInRect` (1545)

38.3.9 PtInRect

Synopsis: Check whether a point is inside a rectangle.

Declaration: `function PtInRect(const Rect: TRect;const p: TPoint) : 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, therefor 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, 0 will be.

See also: `IntersectRect` (1544), `UnionRect` (1546), `IsRectEmpty` (1544), `OffsetRect` (1544), `InflateRect` (1544), `Size` (1545)

38.3.10 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` (1543), `PtInRect` (1545), `IntersectRect` (1544), `UnionRect` (1546), `IsRectEmpty` (1544), `OffsetRect` (1544), `InflateRect` (1544), `Size` (1545)

38.3.11 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` (1545), `IntersectRect` (1544), `IsRectEmpty` (1544), `OffsetRect` (1544), `InflateRect` (1544), `CenterPoint` (1543), `IsRectEmpty` (1544)

38.3.12 UnionRect

Synopsis: Return the union of 2 rectangles.

Declaration: `function UnionRect (var Rect: TRect; const R1: TRect; const R2: TRect)
: Boolean`

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` (1545), `IntersectRect` (1544), `IsRectEmpty` (1544), `OffsetRect` (1544), `InflateRect` (1544), `Size` (1545)

38.4 IClassFactory

38.4.1 Description

`IClassFactory` is defined for Delphi/Kylix compatibility and should not be used.

38.4.2 Method overview

Page	Property	Description
1546	<code>CreateInstance</code>	Create a new instance of an interface.
1546	<code>LockServer</code>	Lock ActiveX server object.

38.4.3 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.

38.4.4 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.

38.5 ISequentialStream

38.5.1 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` ([1547](#))

38.5.2 Method overview

Page	Property	Description
1547	Read	Read data from the stream
1547	Write	Write data to the stream

38.5.3 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` ([1547](#))

38.5.4 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` ([1547](#))

38.6 IStream

38.6.1 Description

An abstract interface for an external (non pascal) stream, as defined in Microsoft COM interfaces

38.6.2 Method overview

Page	Property	Description
1550	Clone	Clone the stream instance
1549	Commit	Commit data to the stream
1548	CopyTo	Copy data from one stream to another
1549	LockRegion	Lock a region of bytes in the stream
1549	Revert	Revert changes
1548	Seek	Set the stream position
1548	SetSize	Set the stream size
1550	Stat	return information about the stream.
1549	UnlockRegion	Unlocks a previously locked region of bytes in the stream

38.6.3 IStream.Seek

Synopsis: Set the stream position

Declaration: `function Seek(dlibMove: Largeint; dwOrigin: LongInt;
out libNewPosition: Largeint) : 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.

38.6.4 IStream.SetSize

Synopsis: Set the stream size

Declaration: `function SetSize(libNewSize: Largeint) : 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.

38.6.5 IStream.CopyTo

Synopsis: Copy data from one stream to another

Declaration: `function CopyTo(stm: IStream; cb: Largeint; out cbRead: Largeint;
out cbWritten: Largeint) : 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.

38.6.6 IStream.Commit

Synopsis: Commit data to the stream

Declaration: `function Commit(grfCommitFlags: LongInt) : 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.

38.6.7 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.

38.6.8 IStream.LockRegion

Synopsis: Lock a region of bytes in the stream

Declaration: `function LockRegion(libOffset: Largeint;cb: Largeint;
dwLockType: LongInt) : 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.

38.6.9 IStream.UnlockRegion

Synopsis: Unlocks a previously locked region of bytes in the stream

Declaration: `function UnlockRegion(libOffset: Largeint;cb: Largeint;
dwLockType: LongInt) : 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.

38.6.10 IStream.Stat

Synopsis: return information about the stream.

Declaration: `function Stat(out statstg: TStatStg; grfStatFlag: LongInt) : 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.

38.6.11 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.

Chapter 39

Reference for unit 'typinfo'

39.1 Used units

Table 39.1: Used units by unit 'typinfo'

Name	Page
System	1118
sysutils	1360

39.2 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 5 unit with the same name. The only calls that are still missing are the Variant calls, since Free Pascal does not support the variant type yet.

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`.

39.3 Auxiliary functions

Other `typinfo` related functions.

Table 39.2:

Name	Description
GetEnumName (1562)	Get an enumerated type element name
GetEnumValue (1564)	Get ordinal number of an enumerated type, based on the name.
GetEnumNameCount (1563)	Get number of elements in an enumerated type.
GetTypeData (1576)	Skip type name and return a pointer to the type data
SetToString (1586)	Convert a set to its string representation
StringToSet (1588)	Convert a string representation of a set to a set

39.4 Getting or setting property values

Functions to set or set a property's value.

Table 39.3:

Name	Description
GetEnumProp (1563)	Return the value of an enumerated type property
GetFloatProp (1564)	Return the value of a float property
GetInt64Prop (1565)	Return the value of an Int64 property
GetMethodProp (1566)	Return the value of a procedural type property
GetObjectProp (1568)	Return the value of an object property
GetOrdProp (1570)	Return the value of an ordinal type property
GetProperty (1573)	Return the value of a property as a variant
GetSetProp (1574)	Return the value of a set property
GetStrProp (1575)	Return the value of a string property
GetWideStrProp (1577)	Return the value of a widestring property
GetVariantProp (1577)	Return the value of a variant property
SetEnumProp (1581)	Set the value of an enumerated type property
SetFloatProp (1581)	Set the value of a float property
SetInt64Prop (1582)	Set the value of an Int64 property
SetMethodProp (1583)	Set the value of a procedural type property
SetObjectProp (1583)	Set the value of an object property
SetOrdProp (1584)	Set the value of an ordinal type property
SetPropValue (1584)	Set the value of a property through a variant
SetSetProp (1585)	Set the value of a set property
SetStrProp (1585)	Set the value of a string property
SetWideStrProp (1587)	Set the value of a widestring property
SetVariantProp (1587)	Set the value of a variant property

39.5 Examining published property information

Functions for retrieving or examining property information

Table 39.4:

Name	Description
FindPropInfo (1561)	Getting property type information, With error checking.
GetPropInfo (1571)	Getting property type information, No error checking.
GetPropInfos (1571)	Find property information of a certain kind
GetObjectPropClass (1569)	Return the declared class of an object property
GetPropList (1572)	Get a list of all published properties
IsPublishedProp (1578)	Is a property published
IsStoredProp (1578)	Is a property stored
PropIsType (1579)	Is a property of a certain kind
PropType (1580)	Return the type of a property

39.6 Constants, types and variables

39.6.1 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 (1573) 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 (1577) 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 (1584) 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 (1577) function.

```
ptConst = 3
```

Constant used in acces method

```
ptField = 0
```

Property acces directly from field

```
ptStatic = 1
```

Property acces via static method

```
ptVirtual = 2
```

Property acces via virtual method

```
tkAny = [ (TTypeKind) .. (TTypeKind) ]
```

Any property type

```
tkMethods = [tkMethod]
```

Only method properties. (event handlers)

```
tkProcedure = tkProcVar
```

Procedure kind

```
tkProperties = tkAny - tkMethods - [tkUnknown]
```

Real properties. (not methods)

```
tkString = tkSSString
```

Alias for the `tsSSString` enumeration value

39.6.2 Types

```
PManagedField = ^TManagedField
```

`PManagedField` is a pointer to `TManagedField` ([1557](#)). It is used to describe automatically managed fields in records when the type kind is `tkRecord`.

```
PProcedureParam = ^TProcedureParam
```

`PProcedureParam` is a pointer to `TProcedureParam`. It is used in `TProcedureSignature` ([1558](#)).

```
PPropInfo = ^TPropInfo
```

Pointer to `TPropInfo` ([1559](#)) record

```
PPropList = ^TPropList
```

Pointer to `TPropList` ([1559](#))

```
PTypeInfo = ^PTypeInfo
```

Pointer to PTypeInfo (1555) pointer

```
PTypeData = ^TTypeData
```

Pointer to TTypeData (1559) record.

```
PTypeInfo = ^TTypeInfo
```

Pointer to TTypeInfo (1559) record

```
PVmtFieldEntry = ^TVmtFieldEntry
```

Pointer to #rtl.typinfo.TVmtFieldEntry (1560) type.

```
PVmtFieldTable = ^TVmtFieldTable
```

Pointer to #rtl.typinfo.TVmtFieldTable (1561) type.

```
ShortStringBase = string
```

ShortStringBase is the base definition of a short string.

```
TArrayTypeData = packed record
  Size : SizeInt;
  ElCount : SizeInt;
  ElType : PTypeInfo;
  DimCount : Byte;
  Dims : 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`).

```
TCallConv = (ccReg, ccCdecl, ccPascal, ccStdCall, ccSafeCall, ccCppdecl,
  ccFar16, ccOldFPCCall, ccInternProc, ccSysCall, ccSoftFloat,
  ccMWPascal)
```

Table 39.5: Enumeration values for type TCallConv

Value	Explanation
<code>ccCdecl</code>	Cdecl calling convention.
<code>ccCppdecl</code>	Cppdecl calling convention
<code>ccFar16</code>	Far16 calling convention (Delphi compatibility)
<code>ccInternProc</code>	InternProc calling convention (compiler internal)
<code>ccMWPascal</code>	MWPascal (MetroWerks Pascal) calling convention.
<code>ccOldFPCCall</code>	OldFPCCall calling convention (deprecated)
<code>ccPascal</code>	Pascal calling convention.
<code>ccReg</code>	Register calling convention
<code>ccSafeCall</code>	SafeCall calling convention.
<code>ccSoftFloat</code>	Softfloat calling convention.
<code>ccStdCall</code>	stdcall calling convention.
<code>ccSysCall</code>	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 39.6: 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; const PropName: string;
                        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`.

```
TIntfFlag = (ifHasGuid, ifDispInterface, ifDispatch, ifHasStrGUID)
```

Table 39.7: 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 ([1556](#)).

```
TIntfFlagsBase = Set of TIntfFlag
```

Set of TIntfFlag ([1556](#)).

```
TManagedField = packed record
  TypeRef : 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.

```
TMethodKind = (mkProcedure, mkFunction, mkConstructor, mkDestructor,
  mkClassProcedure, mkClassFunction, mkClassConstructor,
  mkClassDestructor, mkOperatorOverload)
```

Table 39.8: 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 Desctructor
mkFunction	Function method
mkOperatorOverload	Operator overloader
mkProcedure	Procedure method.

Method type description

```
TOrdType = (otSByte, otUByte, otSWord, otUWord, otSLong, otULong)
```

Table 39.9: Enumeration values for type TOrdType

Value	Explanation
otSByte	Signed byte
otSLong	Signed longint
otSWord	Signed word
otUByte	Unsigned byte
otULong	Unsigned longing (Cardinal)
otUWord	Unsigned word

If the property is and ordinal type, then TOrdType determines the size and sign of the ordinal type:

```
TParamFlag = (pfVar, pfConst, pfArray, pfAddress, pfReference, pfOut)
```

Table 39.10: 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)
pfOut	Parameter is a string parameter
pfReference	Parameter is passed by reference
pfVar	Parameter is a var parameter (passed by reference)

TParamFlag describes a parameter.

TParamFlags = Set of TParamFlag

The kind of parameter for a method

```
TProcedureParam = packed record
  Flags : Byte;
  ParamType : PTypeInfo;
  Name : ShortString;
end
```

TProcedureParam describes a single parameter to a procedure (or function).

```
TProcedureSignature = packed record
  Flags : Byte;
  CC : TCallConv;
  ResultType : 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 ([1558](#))).

TProcInfoProc = procedure(PropInfo: PPropInfo) of object

Property info callback method

```
TPropData = packed record
  PropCount : Word;
  PropList : record
    _alignmentdummy : PtrInt;
  end;
end
```

The TPropData record is not used, but is provided for completeness and compatibility with Delphi.

```

TPropInfo = packed record
  PropType : PTypeInfo;
  GetProc : CodePointer;
  SetProc : CodePointer;
  StoredProc : CodePointer;
  Index : Integer;
  Default : LongInt;
  NameIndex : SmallInt;
  PropProcs : Byte;
  Name : ShortString;
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.

```
TPropList = Array[0..65535] of PPropInfo
```

Array of property information pointers

```

TSetPropValue = procedure (Instance: TObject; const PropName: string;
                           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.

```

TTypeInfoData = packed record
end

```

If the typeinfo kind is tkClass, then the property information follows the UnitName string, as an array of TPropInfo (1559) records.

```

TTypeInfo = record
  Kind : TTypeInfoKind;
  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 TTypeInfoData (1559) record.

```

TTypeInfoKind = (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 39.11: Enumeration values for type TTypeKind

Value	Explanation
tkArray	Array property.
tkAString	Ansistring property.
tkBool	Boolean property.
tkChar	Char property.
tkClass	Class property.
tkClassRef	Class of type
tkDynArray	Dynamical array property.
tkEnumeration	Enumeration type property.
tkFile	File type (both text and binary)
tkFloat	Float property.
tkHelper	Helper class type.
tkInt64	Int64 property.
tkInteger	Integer property.
tkInterface	Interface property.
tkInterfaceRaw	Raw interface property.
tkLString	Longstring property.
tkMethod	Method property.
tkObject	Object property.
tkPointer	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.

```
TTypeKinds = Set of TTypeKind
```

Set of TTypeKind (1560) enumeration.

```
TVmtFieldEntry = packed record
  FieldOffset : PtrUInt;
  TypeIndex : Word;
  Name : ShortString;
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.

```
TVmtFieldTable = packed record
```

```

Count : Word;
ClassTab : Pointer;
Fields : Array[0..0] of TVmtFieldEntry;
end

```

TVmtFieldTable describes the fields for which RTTI was generated. A TVmtFieldTable entry is generated by the compiler in the RTI information, it is not something one creates manually. Basically it contains a list of TVmtFieldEntry (1560) values.

39.7 Procedures and functions

39.7.1 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 (1571) 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 (1571), GetPropList (1572), GetPropInfos (1571)

Listing: ./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.

```

39.7.2 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 lower-cased, but this may change in the future.

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` ([1570](#)), `GetEnumValue` ([1564](#))

Listing: `./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;

```

```

WriteIn ( 'GetEnumName           : ',GetEnumName( TI ,Ord(O. MyEnumField))) ;
WriteIn ( 'GetEnumValue( mefirst ) : ',GetEnumName( TI ,GetEnumValue( TI , ' mefirst '))) ;
O. Free ;
end .

```

39.7.3 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` ([1564](#)), `GetEnumName` ([1562](#))

39.7.4 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 a 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` ([1581](#)), `GetOrdProp` ([1570](#)), `GetStrProp` ([1575](#)), `GetInt64Prop` ([1565](#)), `GetMethodProp` ([1566](#)), `GetSetProp` ([1574](#)), `GetObjectProp` ([1568](#)), `GetEnumProp` ([1563](#))

Listing: `./typinfex/ex2.pp`

```

program example2 ;

{ This program demonstrates the GetEnumProp function }

{$mode objfpc}

uses rttiobj , 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.

```

39.7.5 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` (1562).

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` (1562), `SetOrdProp` (1584)

39.7.6 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` (1581), `GetOrdProp` (1570), `GetStrProp` (1575), `GetInt64Prop` (1565), `GetMethodProp` (1566), `GetSetProp` (1574), `GetObjectProp` (1568), `GetEnumProp` (1563)

Listing: `./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.

```

39.7.7 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

Description: 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` (1582), `GetOrdProp` (1570), `GetStrProp` (1575), `GetFloatProp` (1564), `GetMethodProp` (1566), `GetSetProp` (1574), `GetObjectProp` (1568), `GetEnumProp` (1563)

Listing: ./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.

```

39.7.8 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` ([1582](#)), `GetOrdProp` ([1570](#)), `GetStrProp` ([1575](#)), `GetFloatProp` ([1564](#)), `GetInt64Prop` ([1565](#)), `GetSetProp` ([1574](#)), `GetObjectProp` ([1568](#)), `GetEnumProp` ([1563](#))

39.7.9 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` (1583), `GetOrdProp` (1570), `GetStrProp` (1575), `GetFloatProp` (1564), `GetInt64Prop` (1565), `GetSetProp` (1574), `GetObjectProp` (1568), `GetEnumProp` (1563)

Listing: `./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 address (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:=GetPropertyInfo(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.

```

39.7.10 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` (1583), `GetOrdProp` (1570), `GetStrProp` (1575), `GetFloatProp` (1564), `GetInt64Prop` (1565), `GetSetProp` (1574), `GetObjectProp` (1568), `GetEnumProp` (1563)

Listing: `./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, TObj) 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.

39.7.11 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` (1568).

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` (1583), `GetOrdProp` (1570), `GetStrProp` (1575), `GetFloatProp` (1564), `GetInt64Prop` (1565)

39.7.12 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 typecasted 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` (1584), `GetStrProp` (1575), `GetFloatProp` (1564), `GetInt64Prop` (1565), `GetMethodProp` (1566), `GetSetProp` (1574), `GetObjectProp` (1568), `GetEnumProp` (1563)

Listing: `./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.

```

39.7.13 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` ([1571](#)), `GetPropList` ([1572](#))

39.7.14 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` (1571), `GetPropList` (1572)

Listing: `./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 : ', TypeName[typinfo.PropType(O, Name)]);`

end;

end;

`FreeMem(PP);`

`O.Free;`

end.

39.7.15 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 ([1571](#)), GetPropInfo ([1571](#))

Listing: ./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;

pri : 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.

39.7.16 GetPropValue

Synopsis: Get property value as a string.

Declaration: `function GetPropValue(Instance: TObject;const PropName: string)
: Variant
function GetPropValue(Instance: TObject;const PropName: string;
PreferStrings: Boolean) : Variant`

Visibility: default

Description: Due to missing Variant support, GetPropValue is not yet implemented. The declaration is provided for compatibility with Delphi.

39.7.17 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 (1566) function, no reference counting is done.

Errors: If the property PropName does not exist, an EPropertyError exception is raised.

See also: GetInterfaceProp (1566), SetRawInterfaceProp (1584)

39.7.18 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 it's 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 (1586). 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 (1585), GetStrProp (1575), GetFloatProp (1564), GetInt64Prop (1565), GetMethodProp (1566)

Listing: ./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;

end;

Var
  S : TMyEnums;

begin
  O := TMyTestObject.Create;
  O.SetField := [mefirst, meSecond, meThird];
  Writeln ('Set property      : ');
  Writeln ('Value                               : ', SetAsString(O.SetField));
  Writeln ('Ord(Value)                           : ', Longint(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', Integer(S));
  Write ('Set (name, [mefirst, methird]) : ');
  Writeln (SetAsString(O.SetField));
  S := [meSecond];
  SetOrdProp(O, PI, Integer(S));
  Write ('Set (propinfo, [meSecond]) : ');
  Writeln (SetAsString(O.SetField));
  O.Free;
end.

```

39.7.19 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` (1585), `SetWideStrProp` (1587), `GetOrdProp` (1570), `GetFloatProp` (1564), `GetInt64Prop` (1565), `GetMethodProp` (1566)

Listing: `./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'));
  Writeln('Get (propinfo)        : ', GetStrProp(O, PI));
  SetStrProp(O, 'AnsiStringField', 'First');
  Writeln('Set (name, ''First'')   : ', O.AnsiStringField);
  SetStrProp(O, PI, 'Second');
  Writeln('Set (propinfo, ''Second'') : ', O.AnsiStringField);
  O.Free;
end.
```

39.7.20 GetTypeData

Synopsis: Return a pointer to type data, based on type information.

Declaration: `function GetTypeData(TypeInfo: PTypeInfo) : PTypeData`

Visibility: default

Description: `GetTypeData` returns a pointer to the `TTypeData` record that follows after the `TTypeInfo` record pointed to by `TypeInfo`. It essentially skips the `Kind` and `Name` fields in the `TTypeInfo` record.

Errors: None.

39.7.21 GetUnicodeStrProp

Synopsis: Get UnicodeString-valued property

Declaration: `function GetUnicodeStrProp(Instance: TObject; PropInfo: PPropInfo)
: UnicodeString
function GetUnicodeStrProp(Instance: TObject; const PropName: string)
: UnicodeString`

Visibility: default

Description: `GetUnicodeStrProp` returns the `UnicodeString` property from `Instance`, where the property is identified by the `PropInfo` pointer or the `PropertyName`.

Errors: If no property of the indicated name exists, or the value is not a unicode string, an exception will occur.

See also: `GetStrProp` ([1575](#)), `SetUnicodeStrProp` ([1587](#))

39.7.22 GetVariantProp

Synopsis: Return the value of a variant property.

Declaration: `function GetVariantProp(Instance: TObject; PropInfo: PPropInfo) : Variant
function GetVariantProp(Instance: TObject; const PropName: string)
: Variant`

Visibility: default

Description: Due to missing `Variant` support, the `GetVariantProp` function is not yet implemented. Provided for Delphi compatibility only.

See also: `SetVariantProp` ([1587](#))

39.7.23 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` ([1575](#)), `SetWideStrProp` ([1587](#)), `GetOrdProp` ([1570](#)), `GetFloatProp` ([1564](#)), `GetInt64Prop` ([1565](#)), `GetMethodProp` ([1566](#))

Visibility: default

Description: `IsStoredProp` returns `True` if the `Stored` modifier evaluates to `True` for the property described by `PropInfo` or with name `PropName` for object `Instance`. It returns `False` otherwise. If the function returns `True`, this indicates that the property should be written when streaming the object `Instance`.

If there was no `stored` modifier in the declaration of the property, `True` will be returned.

Errors: No checking is done whether `Instance` is non-nil, or whether `PropInfo` describes a valid property of `Instance`. Specifying an invalid property name in `PropName` will result in an `EPropertyError` exception.

See also: `IsPublishedProp` (1578), `PropIsType` (1579)

Listing: `./typinfex/ex11.pp`

```

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.

```

39.7.26 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.

InstanceAn 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` (1578), `IsStoredProp` (1578), `PropType` (1580)

Listing: ./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.

```

39.7.27 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:

AClassA class pointer.

InstanceAn 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` (1578), `IsStoredProp` (1578), `PropIsType` (1579)

Listing: ./typinfex/ex17.pp

```

program example17;

{ This program demonstrates the PropType function }

{$mode objfpc}

uses rttiobj, typinfo;

Var
  O : TMyTestObject;

begin
  O:= TMyTestObject.Create;
  WriteIn( 'Property tests      : ');
  Write( 'PropType(O, BooleanField)      : ');
  WriteIn(TypeNames[PropType(O, 'BooleanField')]);
  Write( 'PropType(Class, BooleanField) : ');
  WriteIn(TypeNames[PropType(O.ClassType, 'BooleanField')]);
  Write( 'PropType(O, ByteField)      : ');
  WriteIn(TypeNames[PropType(O, 'ByteField')]);
  Write( 'PropType(Class, ByteField)    : ');
  WriteIn(TypeNames[PropType(O.ClassType, 'ByteField')]);
  O.Free;
end.

```

39.7.28 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 (1564).

For an example, see GetEnumProp (1563).

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 (1563), SetStrProp (1585), SetFloatProp (1581), SetInt64Prop (1582), SetMethodProp (1583)

39.7.29 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 (1564).

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 (1564), SetOrdProp (1584), SetStrProp (1585), SetInt64Prop (1582), SetMethodProp (1583)

39.7.30 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 (1565).

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 (1565), GetMethodProp (1566), SetOrdProp (1584), SetStrProp (1585), SetFloatProp (1581)

39.7.31 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 (1566), SetObjectProp (1583), SetOrdProp (1584), SetStrProp (1585), SetFloatProp (1581), SetInt64Prop (1582), SetMethodProp (1583)

39.7.32 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` (1566).

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` (1566), `SetOrdProp` (1584), `SetStrProp` (1585), `SetFloatProp` (1581), `SetInt64Prop` (1582)

39.7.33 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` (1568).

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` (1568), `SetOrdProp` (1584), `SetStrProp` (1585), `SetFloatProp` (1581), `SetInt64Prop` (1582), `SetMethodProp` (1583)

39.7.34 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` ([1570](#)).

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` ([1570](#)), `SetStrProp` ([1585](#)), `SetFloatProp` ([1581](#)), `SetInt64Prop` ([1582](#)), `SetMethodProp` ([1583](#))

39.7.35 SetPropValue

Synopsis: Set property value as variant

Declaration: `procedure SetPropValue(Instance: TObject; const PropName: string;`
`const Value: Variant)`

Visibility: default

Description: Due to missing Variant support, this function is not yet implemented; it is provided for Delphi compatibility only.

39.7.36 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` ([1582](#)) procedure, no reference counting is done.

Errors: If the property `PropName` does not exist, an `EPropertyError` exception is raised.

See also: `SetInterfaceProp` ([1582](#)), `GetRawInterfaceProp` ([1574](#))

39.7.37 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` ([1588](#)) function.

The value can be formed using the `SetToString` ([1586](#)) function.

For an example, see `GetSetProp` ([1574](#)).

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` ([1574](#)), `SetOrdProp` ([1584](#)), `SetStrProp` ([1585](#)), `SetFloatProp` ([1581](#)), `SetInt64Prop` ([1582](#)), `SetMethodProp` ([1583](#)), `SetToString` ([1586](#)), `StringToSet` ([1588](#))

39.7.38 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` ([1575](#))

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` ([1575](#)), `SetWideStrProp` ([1587](#)), `SetOrdProp` ([1584](#)), `SetFloatProp` ([1581](#)), `SetInt64Prop` ([1582](#)), `SetMethodProp` ([1583](#))

39.7.39 SetToString

Synopsis: Convert set to a string description

```

Declaration: function SetToString (TypeInfo: PTypeInfo; Value: Integer;
                                Brackets: Boolean) : string
function SetToString (PropInfo: PPropInfo; Value: Integer;
                    Brackets: Boolean) : string
function SetToString (PropInfo: PPropInfo; Value: Integer) : 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 (1562), GetEnumValue (1564), StringToSet (1588)

Listing: ./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, '[messecond, methird]');
  SetOrdProp(O, PI, I);
  I := GetOrdProp(O, PI);
  Writeln('Value  : ', SetToString(PI, I, True));
  O.Free;
end.

```

39.7.40 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 ([1585](#)), GetUnicodeStrProp ([1577](#))

39.7.41 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.

39.7.42 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 ([1577](#)), SetStrProp ([1585](#)), SetOrdProp ([1584](#)), SetFloatProp ([1581](#)), SetInt64Prop ([1582](#)), SetMethodProp ([1583](#))

39.7.43 StringToSet

Synopsis: Convert string description to a set.

Declaration: `function StringToSet (PropInfo: PPropInfo; const Value: string) : Integer`
`function StringToSet (TypeInfo: PTypeInfo; const Value: string) : Integer`

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` ([1564](#)) function.

For an example, see `SetToString` ([1586](#)).

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` ([1562](#)), `GetEnumValue` ([1564](#)), `SetToString` ([1586](#))

39.8 EPropertyConvertError

39.8.1 Description

`EPropertyConvertError` is not used in the Free Pascal implementation of the `typinfo` unit, but is declared for Delphi compatibility.

39.9 EPropertyError

39.9.1 Description

Exception raised in case of an error in one of the functions.

Chapter 40

Reference for unit 'Unix'

40.1 Used units

Table 40.1: Used units by unit 'Unix'

Name	Page
BaseUnix	100
System	1118
unixtype	1623

40.2 Constants, types and variables

40.2.1 Constants

`ARG_MAX = UnixType . ARG_MAX`

Maximum number of arguments to a program.

`fs_ext = $137d`

File system type (TStatFS ([1604](#))): (ext) Extended

`fs_ext2 = $ef53`

File system type (TStatFS ([1604](#))): (ext2) Second extended

`fs_iso = $9660`

File system type (TStatFS ([1604](#))): ISO 9660

`fs_minix = $137f`

File system type (TStatFS ([1604](#))): Minix

`fs_minix_30 = $138f`

File system type (TStatFS (1604)): Minix 3.0

`fs_minix_V2 = $2468`

File system type (TStatFS (1604)): Minix V2

`fs_msdos = $4d44`

File system type (TStatFS (1604)): MSDOS (FAT)

`fs_nfs = $6969`

File system type (TStatFS (1604)): NFS

`fs_old_ext2 = $ef51`

File system type (TStatFS (1604)): (ext2) Old second extended

`fs_proc = $9fa0`

File system type (TStatFS (1604)): PROC fs

`fs_xia = $012FD16D`

File system type (TStatFS (1604)): XIA

`IOctl_TCGETS = $5401`

IOCTL call number: get Terminal Control settings

`LOCK_EX = 2`

FpFlock (1613) Exclusive lock

`LOCK_NB = 4`

FpFlock (1613) Non-blocking operation

`LOCK_SH = 1`

FpFlock (1613) Shared lock

`LOCK_UN = 8`

FpFlock (1613) unlock

`MAP_FAILED = baseunix . MAP_FAILED`

Error return value for mmap: mmap operation failed.

`MAP_FIXED = baseunix . MAP_FIXED`

#rtl.baseunix.FpMMap (163) map type: Interpret addr exactly

MAP_PRIVATE = baseunix . MAP_PRIVATE

#rtl.baseunix.FpMMap (163) map type: Changes are private

MAP_SHARED = baseunix . MAP_SHARED

#rtl.baseunix.FpMMap (163) map type: Share changes

MAP_TYPE = baseunix . MAP_TYPE

#rtl.baseunix.FpMMap (163) map type: Bitmask for type of mapping

MS_ASYNC = 1deprecated

Asynchronous operation flag for msync call

MS_INVALIDATE = 2deprecated

Invalidate other mappings of file flag for msync call

MS_SYNC = 4deprecated

Synchronous operation flag for msync call

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` ([1592](#))

`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` ([157](#)) option: Get process group priority.

`PRIO_PROCESS = UnixType . PRIO_PROCESS`

#rtl.baseunix.fpGetPriority (157) option: Get process priority.

PRIO_USER = UnixType . PRIO_USER

#rtl.baseunix.fpGetPriority (157) option: Get user priority.

PROT_EXEC = baseunix . PROT_EXEC

#rtl.baseunix.FpMMap (163) memory access: page can be executed

PROT_NONE = baseunix . PROT_NONE

#rtl.baseunix.FpMMap (163) memory access: page can not be accessed

PROT_READ = baseunix . PROT_READ

#rtl.baseunix.FpMMap (163) memory access: page can be read

PROT_WRITE = baseunix . PROT_WRITE

#rtl.baseunix.FpMMap (163) 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 (135) record) mode: Block device

STAT_IFCHR = \$2000

File (#rtl.baseunix.stat (135) record) mode: Character device

STAT_IFDIR = \$4000

File (#rtl.baseunix.stat (135) record) mode: Directory

STAT_IFIFO = \$1000

File (#rtl.baseunix.stat (135) record) mode: FIFO

STAT_IFLNK = \$a000

File (#rtl.baseunix.stat (135) record) mode: Link

STAT_IFMT = \$f000

File (#rtl.baseunix.stat (135) record) mode: File type bit mask

STAT_IFREG = \$8000

File (#rtl.baseunix.stat (135) record) mode: Regular file

STAT_IFSOCK = \$c000

File (#rtl.baseunix.stat (135) record) mode: Socket

STAT_IRGRP = STAT_IROTH shl 3

File (#rtl.baseunix.stat (135) record) mode: Group read permission

STAT_IROTH = \$4

File (#rtl.baseunix.stat (135) record) mode: Other read permission

STAT_IRUSR = STAT_IROTH shl 6

File (#rtl.baseunix.stat (135) record) mode: Owner read permission

STAT_IRWXG = STAT_IRWXO shl 3

File (#rtl.baseunix.stat (135) record) mode: Group permission bits mask

STAT_IRWXO = \$7

File (#rtl.baseunix.stat (135) record) mode: Other permission bits mask

STAT_IRWXU = STAT_IRWXO shl 6

File (#rtl.baseunix.stat (135) record) mode: Owner permission bits mask

STAT_ISGID = \$0400

File (#rtl.baseunix.stat (135) record) mode: GID bit set

STAT_ISUID = \$0800

File (#rtl.baseunix.stat (135) record) mode: UID bit set

STAT_ISVTX = \$0200

File (#rtl.baseunix.stat (135) record) mode: Sticky bit set

STAT_IWGRP = STAT_IWOTH shl 3

File (#rtl.baseunix.stat (135) record) mode: Group write permission

STAT_IWOTH = \$2

File (#rtl.baseunix.stat (135) record) mode: Other write permission

STAT_IWUSR = STAT_IWOTH shl 6

File (#rtl.baseunix.stat (135) record) mode: Owner write permission

STAT_IXGRP = STAT_IXOTH shl 3

File (#rtl.baseunix.stat (135) record) mode: Others execute permission

STAT_IXOTH = \$1

File (#rtl.baseunix.stat (135) record) mode: Others execute permission

STAT_IXUSR = STAT_IXOTH shl 6

File (#rtl.baseunix.stat (135) record) mode: Others execute permission

SYS_NMLN = UnixType . SYS_NMLN

Max system name length.

Wait_Any = -1

#rtl.baseunix.fpWaitPID (192): Wait on any process

Wait_Clone = \$80000000

#rtl.baseunix.fpWaitPID (192): Wait on clone processes only.

Wait_MyPGRP = 0

#rtl.baseunix.fpWaitPID (192): Wait processes from current process group

Wait_NoHang = 1

#rtl.baseunix.fpWaitPID (192): Do not wait

Wait_UnTraced = 2

#rtl.baseunix.fpWaitPID (192): Also report stopped but untraced processes

40.2.2 Types

`cbool = UnixType.cbool`

Boolean type

`cchar = UnixType.cchar`

Alias for `#rtl.UnixType.cchar` ([1625](#))

`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 ([1596](#)).

```
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 ([1596](#)).

```
csshort = UnixType.csshort
```

Short signed integer type

```
cuchar = UnixType.cuchar
```

Alias for #rtl.UnixType.cuchar ([1626](#))

```
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` ([1627](#))

```
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.

```
pbool = UnixType.pbool
```

Pointer to boolean type `cbool` ([1596](#))

```
pcchar = UnixType.pcchar
```

Alias for `#rtl.UnixType.pcchar` (1628)

`pcdouble = UnixType.pcdouble`

Pointer to `cdouble` (124) type.

`pcfloat = UnixType.pcfloating`

Pointer to `cfloat` (124) type.

`pcint = UnixType.pcint`

Pointer to `cInt` (1596) 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` (1603) type.

`pclong = UnixType.pclong`

Pointer to `cLong` (1596) type.

`pclonglong = UnixType.pclonglong`

Pointer to `longlong` type.

`pcschar = UnixType.pcschar`

Pointer to character type `cschar` (1597).

`pcshort = UnixType.pcsshort`

Pointer to `cShort` (1597) type.

`pcsigned = UnixType.pcsigned`

Pointer to signed integer type `csigned` (1597).

```
pcsint = UnixType.pcsint
```

Pointer to signed integer type `csint` (1597)

```
pcsize_t = UnixType.psize_t
```

Pointer to character size type `pcsize_t`.

```
pcslong = UnixType.pcslong
```

Pointer of the signed long `clong` (1597)

```
pcslonglong = UnixType.pcslonglong
```

Pointer to Signed longlong type `clonglong` (1597)

```
pcsshort = UnixType.pcsshort
```

Pointer to short signed integer type `csshort` (1597)

```
pcuchar = UnixType.pcuchar
```

Alias for `#rtl.UnixType.pcuchar` (1630)

```
pcuint = UnixType.pcuint
```

Pointer to `cUInt` (1597) 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` (1598) type.

```
pculonglong = UnixType.pculonglong
```

Unsigned longlong type

`pcunsigned = UnixType.pcunsigned`

Alias for `#rtl.unixtype.pcunsigned` (1630)

`pcushort = UnixType.pcushort`

Pointer to `cuShort` (1598) type.

`pDev = UnixType.pDev`

Pointer to `TDev` (1603) type.

`pGid = UnixType.pGid`

Pointer to `TGid` (1603) type.

`pid_t = UnixType.pid_t`

Process ID type.

`pIno = UnixType.pIno`

Pointer to `TIno` (1603) type.

`pMode = UnixType.pMode`

Pointer to `TMode` (1604) type.

`pnLink = UnixType.pnLink`

Pointer to `TnLink` (1604) type.

`pOff = UnixType.pOff`

Pointer to `TOff` (1604) type.

`pPid = UnixType.pPid`

Pointer to `TPid` (1604) type.

`pSize = UnixType.pSize`

Pointer to `TSize` (1604) type.

`pSize_t = UnixType.pSize_t`

Pointer to type `Size_t`.

`pSocklen = UnixType.pSocklen`

Pointer to TSockLen (1604) type.

```
psSize = UnixType.psSize
```

Pointer to TsSize (1604) 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 (1604) type.

```
ptimespec = UnixType.ptimespec
```

Pointer to timespec (1603) type.

```
ptimeval = UnixType.ptimeval
```

Pointer to timeval (1603) type.

```
ptime_t = UnixType.ptime_t
```

Pointer to time_t (1603) type.

```
pUid = UnixType.pUid
```

Pointer to TUid (1604) 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 (1596) type.

```
TDev = UnixType.TDev
```

Alias for dev_t (1598) type.

```
TFSearchOption = (NoCurrentDirectory, CurrentDirectoryFirst,  
                  CurrentDirectoryLast)
```

Table 40.2: 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 (1615)

```
TGid = UnixType.TGid
```

Alias for gid_t (1598) 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 (1598) type.

```
TIOctlRequest = UnixType.TIOctlRequest
```

Alias for the TIOctlRequest (1635) type in unixtypes

```
TMode = UnixType.TMode
```


Alias for `mode_t` (1598) type.

`TnLink = UnixType.TnLink`

Alias for `nlink_t` (1598) type.

`TOff = UnixType.TOff`

Alias for `off_t` (1598) type.

`TPid = UnixType.TPid`

Alias for `pid_t` (1601) type.

`Tpipe = baseunix.tfilides deprecated`

Array describing a pipe pair of filedescriptors.

`TSize = UnixType.TSize`

Alias for `size_t` (1602) type

`TSocklen = UnixType.TSocklen`

Alias for `socklen_t` (1602) type.

`TsSize = UnixType.TsSize`

Alias for `ssize_t` (1603) type

`tstatfs = UnixType.TStatFs`

`StatFS` returns in `Info` information about the filesystem on which the file `Path` resides. `Info` is of type `TStatFS` (1636).

The function returns zero if the call was succesful, a nonzero value is returned if the call failed.

`TTime = UnixType.TTime`

Alias for `TTime` (1604) type.

`Ttimespec = UnixType.Ttimespec`

Alias for `TimeSpec` (1603) type.

`TTimeVal = UnixType.TTimeVal`

Alias for `timeval` (1603) type.

`TUId = UnixType.TUId`

Alias for `uid_t` (1604) type.

`uid_t = UnixType.uid_t`

User ID type

40.2.3 Variables

`tzdaylight` : Boolean

Indicates whether daylight savings time is active.

`tzname` : Array[boolean] of PChar

Timezone name.

40.3 Procedures and functions

40.3.1 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 succesfully, `False` otherwise.

Errors: In case the function fails and returns `False`, extended error information is returned by the `FpGetErrno` (154) function:

sys_enfile Too many file descriptors for this process.

sys_enfile The system file table is full.

See also: `POpen` (1618), `#rtl.baseunix.FpMkFifo` (163)

Listing: `./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.

```

40.3.2 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 it's standard output can be read from `StreamIn`. Whatever is written by `Prog` to it's 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` ([154](#)) function.

sys_emfile Too many file descriptors for this process.

sys_emfile The system file table is full.

Other errors include the ones by the `fork` and `exec` programs

See also: `AssignPipe` ([1605](#)), `POpen` ([1618](#))

Listing: `./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,'./ex38',[ '-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 ioresult <> 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
end.

```

40.3.3 FpExecL

Synopsis: Execute process (using argument list, environment)

Declaration: `function FpExecL(const PathName: AnsiString;
const S: Array of AnsiString) : 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` (154) function:

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

sys_epermThe file system is mounted *noexec*.

sys_e2bigArgument list too big.

sys_enoexecThe magic number in the file is incorrect.

sys_enoentThe file does not exist.

sys_enomemNot enough memory for kernel, or to split command line.

sys_enotdirA component of the path is not a directory.

sys_eLOOPThe path contains a circular reference (via symlinks).

See also: `FpExecve` (148), `FpExecv` (1610), `FpExecvp` (1611), `FpExecle` (1608), `FpExeclp` (1609), `FpFork` (151)

Listing: ./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.

40.3.4 FpExecLE

Synopsis: Execute process (using argument list, environment)

Declaration: `function FpExecLE(const PathName: AnsiString;
 const S: Array of AnsiString; 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` (154) function:

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

sys_epermThe file system is mounted *noexec*.

sys_e2bigArgument list too big.

sys_enoexecThe magic number in the file is incorrect.

sys_enoentThe file does not exist.

sys_enomemNot enough memory for kernel, or to split command line.

sys_enotdirA component of the path is not a directory.

sys_eloopThe path contains a circular reference (via symlinks).

See also: `FpExecve` (148), `FpExecv` (1610), `FpExecvp` (1611), `FpExecl` (1607), `FpExeclp` (1609), `FpFork` (151)

Listing: ./unixex/ex11.pp

Program Example11;

{ Program to demonstrate the Execl 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. }
 Execl ('/bin/ls -l', envp);

end.

40.3.5 FpExecLP

Synopsis: Execute process (using argument list, environment; search path)

Declaration: `function FpExecLP(const PathName: AnsiString;
 const S: Array of AnsiString) : 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` (154) function:

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

sys_epermThe file system is mounted *noexec*.

sys_e2bigArgument list too big.

sys_enoexecThe magic number in the file is incorrect.

sys_enoentThe file does not exist.

sys_enomemNot enough memory for kernel, or to split command line.

sys_enotdirA component of the path is not a directory.

sys_eloopThe path contains a circular reference (via symlinks).

See also: `FpExecve` (148), `FpExecv` (1610), `FpExecvp` (1611), `FpExecl` (1608), `FpExeclp` (1607), `FpFork` (151)

Listing: ./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.

40.3.6 FpExecLPE

Synopsis: Execute a program in the path, and pass it an environment

Declaration: function FpExecLPE(const PathName: AnsiString;
 const S: Array of AnsiString;env: PPChar) : cint

Visibility: default

Description: FpExecLPE does the same as FpExecLP (1609), 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 (1609), FpExecLE (1608)

40.3.7 FpExecV

Synopsis: Execute process

Declaration: function FpExecV(const PathName: AnsiString;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 (154) function:

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

sys_epermThe file system is mounted *noexec*.

sys_e2bigArgument list too big.

sys_enoexecThe magic number in the file is incorrect.

sys_enoentThe file does not exist.

sys_enomemNot enough memory for kernel.

sys_enotdirA component of the path is not a directory.

sys_eloopThe path contains a circular reference (via symlinks).

See also: `FpExecve` (148), `FpExecvp` (1611), `FpExecle` (1608), `FpExecl` (1607), `FpExeclp` (1609), `FpFork` (151)

Listing: ./unixex/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.

40.3.8 FpExecVP

Synopsis: Execute process, search path

Declaration: `function FpExecVP(const PathName: AnsiString; 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` (154) function:

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

sys_epermThe file system is mounted *noexec*.

sys_e2bigArgument list too big.

sys_enoexecThe magic number in the file is incorrect.

sys_enoentThe file does not exist.

sys_enomemNot enough memory for kernel.

sys_enotdirA component of the path is not a directory.

sys_eloopThe path contains a circular reference (via symlinks).

See also: `FpExecve` (148), `FpExecv` (1610), `FpExecle` (1608), `FpExecl` (1607), `FpExeclp` (1609), `FpFork` (151)

```

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.

```

40.3.10 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 filedescriptor (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 ([154](#)) function:

See also: #rtl.baseunix.FpFcntl ([149](#)), FSync ([1589](#))

40.3.11 fpfStatFS

Synopsis: Retrieve filesystem information.

Declaration: function fpfStatFS(Fd: cint; Info: pstatfs) : cint

Visibility: default

Description: `fpStatFS` returns in `Info` information about the filesystem on which the open file descriptor `fd` resides. `Info` is of type `tstatfs`. The function returns 0 if the call was succesfull, or an error code if the call failed.

Errors: On error, a non-zero error code is returned

See also: `fpStatFS` (1614), `#rtl.baseunix.fpfStat` (152)

40.3.12 `fpfsync`

Synopsis: Flush cached data to disk

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

Visibility: default

Description: `fpfsync` forces the system to write all paged (in-memory) changes to file descriptor `fd` to disk. If the call was succesful, 0 is returned.

Errors: On error, a nonzero error-code is returned.

40.3.13 `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.

40.3.14 `fpStatFS`

Synopsis: Retrieve filesystem 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 filesystem on which the file or path `Path` resides. `Info` is of type `tstatfs`. The function returns 0 if the call was succesfull, or an error code if the call failed.

Errors: On error, a non-zero error code is returned

See also: `fpFStatFS` (1613), `#rtl.baseunix.fpfStat` (184)

40.3.15 fpSystem

Synopsis: Execute and feed command to system shell

Declaration: `function fpSystem(const Command: string) : cint`
`function fpSystem(const Command: AnsiString) : cint`

Visibility: default

Description: `Shell` 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` (151) or `FpExecve` (148) calls. To convert the return value of `fpSystem` to the real return value use `WEXITSTATUS()` (193).

Errors: Errors are reported in `(fpget)ErrNo` (154)

See also: `POpen` (1618), `FpFork` (151), `FpExecve` (148)

Listing: `./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.

```

40.3.16 FSearch

Synopsis: Search for file in search path.

Declaration: `function FSearch(const path: AnsiString;dirlist: Ansistring;`
`CurrentDirStrategy: TFSearchOption) : AnsiString`
`function FSearch(const path: AnsiString;dirlist: AnsiString)`
`: AnsiString`

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` ([1638](#))

Listing: `./unixex/ex46.pp`

Program `Example46`;

{ Program to demonstrate the FSearch function. }

Uses `BaseUnix` , `Unix` , `Strings` ;

begin

WriteIn ('ls is in : ', FSearch ('ls' , **strpas** (fpGetenv ('PATH'))));

end.

40.3.17 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` ([1616](#))

Listing: `./unixex/ex39.pp`

Program `Example39`;

{ Program to demonstrate the GetDomainName function. }

Uses `Unix` ;

begin

WriteIn ('Domain name of this machine is : ', GetDomainName);

end.

40.3.18 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` ([1616](#))

Listing: `./unixex/ex40.pp`

Program Example40;

{ Program to demonstrate the GetHostName function. }

Uses unix;

begin

WriteLn ('Name of this machine is : ',GetHostName);
end.

40.3.19 GetLocalTimezone

Synopsis: Return local timzeone information

Declaration: `procedure GetLocalTimezone(timer: cint;var leap_correct: cint;
 var leap_hit: cint)
 procedure GetLocalTimezone(timer: cint)`

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` ([1617](#)), `ReadTimezoneFile` ([1619](#))

40.3.20 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` ([1619](#))

40.3.21 PClose

Synopsis: Close file opened with POpen (1618)

Declaration: `function PClose(var F: File) : cint`
`function PClose(var F: text) : cint`

Visibility: default

Description: PClose closes a file opened with POpen (1618). It waits for the command to complete, and then returns the exit status of the command.

For an example, see POpen (1618)

Errors: Extended error information is returned by the FpGetErrno (154) function.

See also: POpen (1618)

40.3.22 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 (1618). The result is the same, but PClose returns the exit status of the command Prog.

Errors: Extended error information is returned by the FpGetErrno (154) function. Errors are essentially those of the Execve, Dup and AssignPipe commands.

See also: AssignPipe (1605), PClose (1618)

Listing: ./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 \*"${*}"\* ');
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.

```

40.3.23 ReadTimezoneFile

Synopsis: Read the timezone file and initialize time routines

Declaration: `procedure ReadTimezoneFile(fn: string)`

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` ([1617](#)), `GetLocalTimezone` ([1617](#))

40.3.24 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` ([1639](#)) 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` ([1412](#)), `Time` ([1507](#)), `Now` ([1475](#)), `EpochToLocal` ([1639](#))

40.3.25 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` (168).

Errors: Extended error information is returned by the `FpGetErrno` (154) function:

See also: `#rtl.baseunix.fpCloseDir` (145), `#rtl.baseunix.fpReadDir` (172), `#rtl.baseunix.fpOpenDir` (168), `TellDir` (1621)

40.3.26 SelectText

Synopsis: Wait for event on text file.

Declaration: `function SelectText(var T: Text; Timeout: ptimeval) : cint`
`function SelectText(var T: Text; Timeout: cint) : cint`

Visibility: default

Description: `SelectText` executes the `FpSelect` (175) call on a file of type `Text`. You can specify a timeout in `Timeout`. The `SelectText` call determines itself whether it should check for read or write, depending on how the file was opened : With `Reset` it is checked for reading, with `Rewrite` and `Append` it is checked for writing.

Errors: See `#rtl.baseunix.FpSelect` (175). `SYS_EBADF` can also mean that the file wasn't opened.

See also: `#rtl.baseunix.FpSelect` (175)

40.3.27 SigRaise

Synopsis: Raise a signal (send to current process)

Declaration: `procedure SigRaise(sig: Integer)`

Visibility: default

Description: `SigRaise` sends a `Sig` signal to the current process.

Errors: None.

See also: `#rtl.baseunix.FpKill` (159), `#rtl.baseunix.FpGetPid` (156)

Listing: `./unixex/ex65.pp`

Program `example64`;

{ Program to demonstrate the SigRaise function. }

uses `Unix`, `BaseUnix`;

Var

`oa, na : PSigActionrec`;

```

Procedure DoSig(sig : Longint);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 }
  // this member is linux only, and afaik even there arcane
  na^.Sa_Restorer:=Nil;
  { $endif }
  if fpSigAction(SigUsrc1,na,oa)<>0 then
    begin
      writeln('Error: ',fpgeterrno);
      halt(1);
    end;
    Writeln('Sending USR1 ( ',sigusr1,') signal to self. ');
    SigRaise(sigusr1);
  end.

```

40.3.28 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` ([168](#)).

See also: `#rtl.baseunix.fpCloseDir` ([145](#)), `#rtl.baseunix.fpReadDir` ([172](#)), `#rtl.baseunix.fpOpenDir` ([168](#)), `SeekDir` ([1620](#))

40.3.29 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` ([192](#)) 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 \(192\)](#), [#rtl.baseunix.WTERMSIG \(194\)](#), [#rtl.baseunix.WSTOPSIG \(194\)](#), [#rtl.baseunix.WIFEXITED \(193\)](#), [WIFSTOPPED \(1622\)](#), [#rtl.baseunix.WIFSIGNALED \(194\)](#), [W_EXITCODE \(1622\)](#), [W_STOPCODE \(1622\)](#), [#rtl.baseunix.WEXITSTATUS \(193\)](#)

40.3.30 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 \(192\)](#).

See also: [#rtl.baseunix.FpWaitPID \(192\)](#), [WaitProcess \(1621\)](#), [#rtl.baseunix.WTERMSIG \(194\)](#), [#rtl.baseunix.WSTOPSIG \(194\)](#), [#rtl.baseunix.WIFEXITED \(193\)](#), [#rtl.baseunix.WIFSIGNALED \(194\)](#), [W_EXITCODE \(1622\)](#), [W_STOPCODE \(1622\)](#), [#rtl.baseunix.WEXITSTATUS \(193\)](#)

40.3.31 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 \(192\)](#), [WaitProcess \(1621\)](#), [#rtl.baseunix.WTERMSIG \(194\)](#), [#rtl.baseunix.WSTOPSIG \(194\)](#), [#rtl.baseunix.WIFEXITED \(193\)](#), [WIFSTOPPED \(1622\)](#), [#rtl.baseunix.WIFSIGNALED \(194\)](#), [W_EXITCODE \(1622\)](#), [W_STOPCODE \(1622\)](#), [#rtl.baseunix.WEXITSTATUS \(193\)](#)

40.3.32 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 \(194\)](#) to return `True`

See also: [#rtl.baseunix.FpWaitPID \(192\)](#), [WaitProcess \(1621\)](#), [#rtl.baseunix.WTERMSIG \(194\)](#), [#rtl.baseunix.WSTOPSIG \(194\)](#), [#rtl.baseunix.WIFEXITED \(193\)](#), [WIFSTOPPED \(1622\)](#), [#rtl.baseunix.WIFSIGNALED \(194\)](#), [W_EXITCODE \(1622\)](#), [#rtl.baseunix.WEXITSTATUS \(193\)](#)

Chapter 41

Reference for unit 'unixtype'

41.1 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.

41.2 Constants, types and variables

41.2.1 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 (157)` option: Get process group priority.

`Prio_Process = 0`

`#rtl.baseunix.fpGetPriority (157)` option: Get process priority.

`Prio_User = 2`

`#rtl.baseunix.fpGetPriority (157)` 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: ?

41.2.2 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)

`longdouble = extended`

Usually translates to an extended, but is CPU dependent.

`clonglong = 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 ([1625](#)).

```
csint = cint32
```

Signed integer

```
cslong = Int64
```

The size is CPU dependent.

```
cslonglong = cint64
```

cslonglong is an alias for clonglong ([1626](#)).

```
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)

```
culonglong = uint64
```

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

```
cunsigned = uint
```

Alias for #rtl.unixtype.cuint ([1626](#))

```
cushort = uint16
```

C type: short unsigned integer (half sized)

```
dev_t = uint64
```

Device descriptor type.

```
gid_t = uint32
```

Group ID type.

```
ino64_t = uint64
```

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 = ushort
```

Kernel device type

```
mbstate_t = record
  __count : cint;
  __value : mbstate_value_t;
end
```

This type should never be used directly.


```
mbstate_value_t = record  
end
```

This type should never be used directly. It is part of the `mbstate_t` (1627) type.

```
mode_t = cint
```

Inode mode type.

```
nlink_t = cuint32
```

Number of links type.

```
off64_t = cint64
```

64-bit offset type.

```
off_t = cint
```

Offset type.

```
pcbool = ^cbool
```

Pointer to boolean type `cbool` (1625)

```
pcchar = ^cchar
```

Pointer to `#rtl.UnixType.cchar` (1625)

```
pcdouble = ^cdouble
```

Pointer to `cdouble` (1625) type.

```
pcfloat = ^cfloat
```

Pointer to `cfloat` (1625) type.

```
pcint = ^cint
```

Pointer to `cInt` (1625) 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 (1634) type.

```
pclong = ^clong
```

Pointer to cLong (1625) type.

```
pclongdouble = ^clongdouble
```

Pointer to the long double type clongdouble (1625)

```
pclonglong = ^clonglong
```

Pointer to longlong type.

```
pcschar = ^cschar
```

Pointer to character type cschar (1626).

```
pcshort = ^cshort
```

Pointer to cShort (1626) type.

```
pcsigned = ^csigned
```

Pointer to signed integer type csigned (1626).

```
pcsint = ^csint
```

Pointer to signed integer type csint (1626)

```
pcslong = ^cslong
```

Pointer of the signed long cslong (1626)

```
pcslonglong = ^cslonglong
```

Pointer to Signed longlong type cslonglong (1626)

```
pcsshort = ^csshort
```

Pointer to short signed integer type csshort (1626)

```
pcuchar = ^cuchar
```

Pointer to #rtl.UnixType.cuchar (1626)

```
pcuint = ^cuint
```

Pointer to cUInt (1626) 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 (1627) type.

```
pculonglong = ^culonglong
```

Unsigned longlong type

```
pcunsigned = ^cunsigned
```

Pointer to #rtl.unixtype.cunsigned (1627)

```
pcushort = ^cushort
```

Pointer to cuShort (1627) type.

```
pDev = ^dev_t
```

Pointer to TDev (1634) type.

```
pGid = ^gid_t
```

Pointer to TGid (1634) type.

```
pid_t = cint
```

Process ID type.

```
pIno = ^ino_t
```

Pointer to TIno (1635) type.

```
pIno64 = ^ino64_t
```

Pointer to ino64_t (1627)

```
pkDev = ^kDev_t
```

Pointer to TkDev (1635) type.

```
pmbstate_t = ^mbstate_t
```

Pointer to mbstate_t (1627) type

```
pMode = ^mode_t
```

Pointer to TMode (1635) type.

```
pnLink = ^nlink_t
```

Pointer to TnLink (1635) type.

```
pOff = ^off_t
```

Pointer to TOff (1635) type.

```
pOff64 = ^off64_t
```

Pointer to off64_t type

```
pPid = ^pid_t
```

Pointer to TPid (1635) type.

```
pSize = ^size_t
```

Pointer to TSize (1635) type.

```
psize_t = pSize
```

Pointer to size_t (1634) type.

```
pSockLen = ^socklen_t
```

Pointer to TSockLen (1636) type.

```
pSSize = ^ssize_t
```

Pointer to TsSize (1636) type

```
PStatFS = ^TStatfs
```

Pointer to TStatFS (1636) type.

```
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.

```
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.

```
pthread_cond_t = record
  __c_lock : _pthread_fastlock;
  __c_waiting : pointer;
  __padding : Array[0..48-1-sizeof(_pthread_fastlock)-sizeof(pointer)-sizeof(clonglong)];
  __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.

```
pthread_key_t = cuint
```

Thread local storage key (opaque)

```
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.

```
pthread_mutex_t = record
  __m_reserved : cint;
  __m_count : cint;
```

```

    __m_owner : pointer;
    __m_kind : cint;
    __m_lock : _pthread_fastlock;
end

```

`_pthread_mutex_t` describes a thread mutex. It should be considered an opaque record, the names of the fields can change anytime.

```

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.

```

pthread_rwlock_t = record
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 (1636) type.

```

ptimespec = ^timespec

```

Pointer to timespec (1634) record.

```

ptimeval = ^timeval

```

Pointer to timeval (1635) record.

```

ptime_t = ^time_t

```

Pointer to time_t (1635) type.

```

pUId = ^uid_t

```

Pointer to TUid (1636) type.

```

pwchar_t = ^wchar_t

```

Pointer to `wchar_t` (1636) type.

```

sched_param = record
  __sched_priority : cint;
end

```

Scheduling parameter description record.

```

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.

```

size_t = cuint64

```

Size specification type.

```

socklen_t = cuint32

```

Socket address length type.

```

ssize_t = cint64

```

Small size type.

```

TClock = clock_t

```

Alias for `clock_t` (1625) type.

```

TDev = dev_t

```

Alias for `dev_t` (1627) type.

```

TGid = gid_t

```

Alias for `gid_t` (1627) type.

```

timespec = record
  tv_sec : time_t;
  tv_nsec : clong;
end

```

Record specifying time interval.

```

timeval = record
  tv_sec : time_t;
  tv_usec : clong;
end

```

Time specification type.

```
time_t = cint64
```

Time span type

```
TIno = ino_t
```

Alias for `ino_t` ([1627](#)) type.

```
TIno64 = ino64_t
```

Alias for `ino64_t` ([1627](#))

```
TIOCtlRequest = cint
```

Opaque type used in `FpIOctl` ([159](#))

```
TkDev = kDev_t
```

Alias for `kDev_t` ([1627](#)) type.

```
TMode = mode_t
```

Alias for `mode_t` ([1628](#)) type.

```
TnLink = nlink_t
```

Alias for `nlink_t` ([1628](#)) type.

```
TOff = off_t
```

Alias for `off_t` ([1628](#)) type.

```
TOff64 = off64_t
```

Alias for `off64_t` type.

```
TPid = pid_t
```

Alias for `pid_t` ([1630](#)) type.

```
TSize = size_t
```

Alias for `size_t` ([1634](#)) type

TSockLen = socklen_t

Alias for socklen_t (1634) type.

TSSize = ssize_t

Alias for ssize_t (1634) type

```
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;
  spare : Array[0..4] of clong;
end
```

Record describing a file system in the unix.fstatfs (1623) call.

TTime = time_t

Alias for TTime (1636) type.

TTimeSpec = timespec

Alias for TimeSpec (1634) type.

TTimeVal = timeval

Alias for TimeVal (1635) record.

TUId = uid_t

Alias for uid_t (1636) type.

uid_t = cuint32

User ID type

wchar_t = cint32

Wide character type.

wint_t = cint32

Wide character size type.

```
_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 42

Reference for unit 'unixutil'

42.1 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 ([1589](#)) or baseunix ([100](#)) units.

Most of the functions described here have cross-platform counterparts in the SysUtils ([1360](#)) unit. It is therefore recommended to use that unit.

42.2 Constants, types and variables

42.2.1 Types

`ComStr = string deprecated`

Command-line string type.

`DirStr = string deprecated`

Filename directory part string type.

`ExtStr = string deprecated`

Filename extension part string type.

`NameStr = string deprecated`

Filename name part string type.

`PathStr = string deprecated`

Filename full path string type.

42.2.2 Variables

`Tzseconds : LongInt`

Seconds west of GMT

42.3 Procedures and functions

42.3.1 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` ([1641](#))

42.3.2 EpochToLocal

Synopsis: Convert epoch time to local time

Declaration: `procedure EpochToLocal(epoch: LongInt; var year: Word; var month: Word;
var day: Word; var hour: Word; var minute: Word;
var second: Word)`

Visibility: default

Description: Converts the epoch time (=Number of seconds since 00:00:00, January 1, 1970, corrected for your time zone) to local date and time.

This function takes into account the timzeone settings of your system.

Errors: None

See also: `LocalToEpoch` ([1641](#))

Listing: `./unutilx/ex3.pp`

Program `Example3;`

`{ Program to demonstrate the EpochToLocal function. }`

Uses `BaseUnix, Unix, UnixUtil;`

Var `Year, month, day, hour, minute, seconds : Word;`

begin

`EpochToLocal (FPTIME, Year, month, day, hour, minute, seconds);`

`Writeln ('Current date : ', Day:2, '/', Month:2, '/', Year:4);`

`Writeln ('Current time : ', Hour:2, ': ', minute:2, ': ', seconds:2);`

end.

42.3.3 GetFS

Synopsis: Return file selector

Declaration: `function GetFS (var T: Text) : LongInt`
`function GetFS (var F: File) : LongInt`

Visibility: default

Description: `GetFS` returns the file selector that the kernel provided for your file. In principle you don't need this file selector. Only for some calls it is needed, such as the `#rtl.baseunix.fpSelect` (175) call or so.

Errors: In case the file was not opened, then -1 is returned.

See also: `#rtl.baseunix.fpSelect` (175)

Listing: `./unutilx/ex34.pp`

Program `Example33;`

{ Program to demonstrate the SelectText function. }

Uses `Unix;`

Var `tv : TimeVal;`

begin

`Writeln ('Press the <ENTER> to continue the program.');`
{ Wait until File descriptor 0 (=Input) changes }
`SelectText (Input, nil);`
{ Get rid of <ENTER> in buffer }

`readln;`

`Writeln ('Press <ENTER> key in less than 2 seconds...');`

`tv.tv_sec:=2;`

`tv.tv_sec:=0;`

`if SelectText (Input, @tv) > 0 then`

`Writeln ('Thank you !')`

`else`

`Writeln ('Too late !');`

`end.`

42.3.4 GregorianToJulian

Synopsis: Converts a gregorian date to a julian date

Declaration: `function GregorianToJulian (Year: LongInt; Month: LongInt; Day: LongInt)`
`: LongInt`

Visibility: default

Description: `GregorianToJulian` takes a gregorian date and converts it to a Julian day.

Errors: None.

See also: `JulianToGregorian` (1641)

42.3.5 JulianToGregorian

Synopsis: Converts a julian date to a gregorian date

Declaration: `procedure JulianToGregorian(JulianDN: LongInt; var Year: Word;
var Month: Word; var Day: Word)`

Visibility: default

Description: `JulianToGregorian` takes a julian day and converts it to a gregorian date. (Start of the Julian Date count is from 0 at 12 noon 1 JAN -4712 (4713 BC),)

Errors: None.

See also: `GregorianToJulian` ([1640](#))

42.3.6 LocalToEpoch

Synopsis: Convert local time to epoch (unix) time

Declaration: `function LocalToEpoch(year: Word; month: Word; day: Word; hour: Word;
minute: Word; second: Word) : LongInt`

Visibility: default

Description: Converts the Local time to epoch time (=Number of seconds since 00:00:00, January 1, 1970).

Errors: None

See also: `EpochToLocal` ([1639](#))

Listing: `./unutilx/ex4.pp`

Program `Example4;`

{ Program to demonstrate the LocalToEpoch function. }

Uses `UnixUtil;`

Var `year, month, day, hour, minute, second : Word;`

begin

`Write ('Year : '); readln (Year);`

`Write ('Month : '); readln (Month);`

`Write ('Day : '); readln (Day);`

`Write ('Hour : '); readln (Hour);`

`Write ('Minute : '); readln (Minute);`

`Write ('Seonds : '); readln (Second);`

`Write ('This is : ');`

`Write (LocalToEpoch (year, month, day, hour, minute, second));`

`Writeln (' seconds past 00:00 1/1/1980 ');`

end.

42.3.7 StringToPPChar

Synopsis: Split string in list of null-terminated strings

```
Declaration: function StringToPPChar(S: PChar; ReserveEntries: Integer) : ppchar
              function StringToPPChar(var S: AnsiString; 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 \(1639\)](#), [#rtl.baseunix.FpExecve \(148\)](#)

Listing: ./unutillex/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 43

Reference for unit 'video'

43.1 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` (1661), 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` (1664) will then cp 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` (1661) function can be used: This will increment an internal counter. As long as the counter differs from zero, calling `UpdateScreen` (1664) will not do anything. The counter can be lowered with `UnlockScreenUpdate` (1664). When it reaches zero, the next call to `UpdateScreen` (1664) 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` (1663) call. The current video driver can be retrieved using the `GetVideoDriver` (1659) 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.

43.2 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: `./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.
```

43.3 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` (1663) function. The various functions that can be implemented are located in the `TVideoDriver` (1653) 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);
GetCursorType   : function : Word;
SetCursorType   : procedure (NewType: Word);
GetCapabilities : 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: `./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 .

```

43.4 Constants, types and variables

43.4.1 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` (1655) 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, iso11, iso12, iso13, iso14, iso15, iso16, iso17, iso18, iso19, iso20, iso21, iso22, iso23, iso24, iso25, iso26, iso27, iso28, iso29, iso30, iso31, iso32, iso33, iso34, iso35, iso36, iso37, iso38, iso39, iso40, iso41, iso42, iso43, iso44, iso45, iso46, iso47, iso48, iso49, iso50, iso51, iso52, iso53, iso54, iso55, iso56, iso57, iso58, iso59, iso60, iso61, iso62, iso63, iso64, iso65, iso66, iso67, iso68, iso69, iso70, iso71, iso72, iso73, iso74, iso75, iso76, iso77, iso78, iso79, iso80, iso81, iso82, iso83, iso84, iso85, iso86, iso87, iso88, iso89, iso90, iso91, iso92, iso93, iso94, iso95, iso96, iso97, iso98, iso99, iso100, iso101, iso102, iso103, iso104, iso105, iso106, iso107, iso108, iso109, iso110, iso111, iso112, iso113, iso114, iso115, iso116, iso117, iso118, iso119, iso120, iso121, iso122, iso123, iso124, iso125, iso126, iso127, iso128, iso129, iso130, iso131, iso132, iso133, iso134, iso135, iso136, iso137, iso138, iso139, iso140, iso141, iso142, iso143, iso144, iso145, iso146, iso147, iso148, iso149, iso150, iso151, iso152, iso153, iso154, iso155, iso156, iso157, iso158, iso159, iso160, iso161, iso162, iso163, iso164, iso165, iso166, iso167, iso168, iso169, iso170, iso171, iso172, iso173, iso174, iso175, iso176, iso177, iso178, iso179, iso180, iso181, iso182, iso183, iso184, iso185, iso186, iso187, iso188, iso189, iso190, iso191, iso192, iso193, iso194, iso195, iso196, iso197, iso198, iso199, iso200, iso201, iso202, iso203, iso204, iso205, iso206, iso207, iso208, iso209, iso210, iso211, iso212, iso213, iso214, iso215, iso216, iso217, iso218, iso219, iso220, iso221, iso222, iso223, iso224, iso225, iso226, iso227, iso228, iso229, iso230, iso231, iso232, iso233, iso234, iso235, iso236, iso237, iso238, iso239, iso240, iso241, iso242, iso243, iso244, iso245, iso246, iso247, iso248, iso249, iso250, iso251, iso252, iso253, iso254, iso255, iso256, iso257, iso258, iso259, iso260, iso261, iso262, iso263, iso264, iso265, iso266, iso267, iso268, iso269, iso270, iso271, iso272, iso273, iso274, iso275, iso276, iso277, iso278, iso279, iso280, iso281, iso282, iso283, iso284, iso285, iso286, iso287, iso288, iso289, iso290, iso291, iso292, iso293, iso294, iso295, iso296, iso297, iso298, iso299, iso300, iso301, iso302, iso303, iso304, iso305, iso306, iso307, iso308, iso309, iso310, iso311, iso312, iso313, iso314, iso315, iso316, iso317, iso318, iso319, iso320, iso321, iso322, iso323, iso324, iso325, iso326, iso327, iso328, iso329, iso330, iso331, iso332, iso333, iso334, iso335, iso336, iso337, iso338, iso339, iso340, iso341, iso342, iso343, iso344, iso345, iso346, iso347, iso348, iso349, iso350, iso351, iso352, iso353, iso354, iso355, iso356, iso357, iso358, iso359, iso360, iso361, iso362, iso363, iso364, iso365, iso366, iso367, iso368, iso369, iso370, iso371, iso372, iso373, iso374, iso375, iso376, iso377, iso378, iso379, iso380, iso381, iso382, iso383, iso384, iso385, iso386, iso387, iso388, iso389, iso390, iso391, iso392, iso393, iso394, iso395, iso396, iso397, iso398, iso399, iso400, iso401, iso402, iso403, iso404, iso405, iso406, iso407, iso408, iso409, iso410, iso411, iso412, iso413, iso414, iso415, iso416, iso417, iso418, iso419, iso420, iso421, iso422, iso423, iso424, iso425, iso426, iso427, iso428, iso429, iso430, iso431, iso432, iso433, iso434, iso435, iso436, iso437, iso438, iso439, iso440, iso441, iso442, iso443, iso444, iso445, iso446, iso447, iso448, iso449, iso450, iso451, iso452, iso453, iso454, iso455, iso456, iso457, iso458, iso459, iso460, iso461, iso462, iso463, iso464, iso465, iso466, iso467, iso468, iso469, iso470, iso471, iso472, iso473, iso474, iso475, iso476, iso477, iso478, iso479, iso480, iso481, iso482, iso483, iso484, iso485, iso486, iso487, iso488, iso489, iso490, iso491, iso492, iso493, iso494, iso495, iso496, iso497, iso498, iso499, iso500, iso501, iso502, iso503, iso504, iso505, iso506, iso507, iso508, iso509, iso510, iso511, iso512, iso513, iso514, iso515, iso516, iso517, iso518, iso519, iso520, iso521, iso522, iso523, iso524, iso525, iso526, iso527, iso528, iso529, iso530, iso531, iso532, iso533, iso534, iso535, iso536, iso537, iso538, iso539, iso540, iso541, iso542, iso543, iso544, iso545, iso546, iso547, iso548, iso549, iso550, iso551, iso552, iso553, iso554, iso555, iso556, iso557, iso558, iso559, iso560, iso561, iso562, iso563, iso564, iso565, iso566, iso567, iso568, iso569, iso570, iso571, iso572, iso573, iso574, iso575, iso576, iso577, iso578, iso579, iso580, iso581, iso582, iso583, iso584, iso585, iso586, iso587, iso588, iso589, iso590, iso591, iso592, iso593, iso594, iso595, iso596, iso597, iso598, iso599, iso600, iso601, iso602, iso603, iso604, iso605, iso606, iso607, iso608, iso609, iso610, iso611, iso612, iso613, iso614, iso615, iso616, iso617, iso618, iso619, iso620, iso621, iso622, iso623, iso624, iso625, iso626, iso627, iso628, iso629, iso630, iso631, iso632, iso633, iso634, iso635, iso636, iso637, iso638, iso639, iso640, iso641, iso642, iso643, iso644, iso645, iso646, iso647, iso648, iso649, iso650, iso651, iso652, iso653, iso654, iso655, iso656, iso657, iso658, iso659, iso660, iso661, iso662, iso663, iso664, iso665, iso666, iso667, iso668, iso669, iso670, iso671, iso672, iso673, iso674, iso675, iso676, iso677, iso678, iso679, iso680, iso681, iso682, iso683, iso684, iso685, iso686, iso687, iso688, iso689, iso690, iso691, iso692, iso693, iso694, iso695, iso696, iso697, iso698, iso699, iso700, iso701, iso702, iso703, iso704, iso705, iso706, iso707, iso708, iso709, iso710, iso711, iso712, iso713, iso714, iso715, iso716, iso717, iso718, iso719, iso720, iso721, iso722, iso723, iso724, iso725, iso726, iso727, iso728, iso729, iso730, iso731, iso732, iso733, iso734, iso735, iso736, iso737, iso738, iso739, iso740, iso741, iso742, iso743, iso744, iso745, iso746, iso747, iso748, iso749, iso750, iso751, iso752, iso753, iso754, iso755, iso756, iso757, iso758, iso759, iso760, iso761, iso762, iso763, iso764, iso765, iso766, iso767, iso768, iso769, iso770, iso771, iso772, iso773, iso774, iso775, iso776, iso777, iso778, iso779, iso780, iso781, iso782, iso783, iso784, iso785, iso786, iso787, iso788, iso789, iso790, iso791, iso792, iso793, iso794, iso795, iso796, iso797, iso798, iso799, iso800, iso801, iso802, iso803, iso804, iso805, iso806, iso807, iso808, iso809, iso810, iso811, iso812, iso813, iso814, iso815, iso816, iso817, iso818, iso819, iso820, iso821, iso822, iso823, iso824, iso825, iso826, iso827, iso828, iso829, iso830, iso831, iso832, iso833, iso834, iso835, iso836, iso837, iso838, iso839, iso840, iso841, iso842, iso843, iso844, iso845, iso846, iso847, iso848, iso849, iso850, iso851, iso852, iso853, iso854, iso855, iso856, iso857, iso858, iso859, iso860, iso861, iso862, iso863, iso864, iso865, iso866, iso867, iso868, iso869, iso870, iso871, iso872, iso873, iso874, iso875, iso876, iso877, iso878, iso879, iso880, iso881, iso882, iso883, iso884, iso885, iso886, iso887, iso888, iso889, iso890, iso891, iso892, iso893, iso894, iso895, iso896, iso897, iso898, iso899, iso900, iso901, iso902, iso903, iso904, iso905, iso906, iso907, iso908, iso909, iso910, iso911, iso912, iso913, iso914, iso915, iso916, iso917, iso918, iso919, iso920, iso921, iso922, iso923, iso924, iso925, iso926, iso927, iso928, iso929, iso930, iso931, iso932, iso933, iso934, iso935, iso936, iso937, iso938, iso939, iso940, iso941, iso942, iso943, iso944, iso945, iso946, iso947, iso948, iso949, iso950, iso951, iso952, iso953, iso954, iso955, iso956, iso957, iso958, iso959, iso960, iso961, iso962, iso963, iso964, iso965, iso966, iso967, iso968, iso969, iso970, iso971, iso972, iso973, iso974, iso975, iso976, iso977, iso978, iso979, iso980, iso981, iso982, iso983, iso984, iso985, iso986, iso987, iso988, iso989, iso990, iso991, iso992, iso993, iso994, iso995, iso996, iso997, iso998, iso999]
```

`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

43.4.2 Types

`PVideoBuf` = `^TVideoBuf`

Pointer type to `TVideoBuf` ([1653](#))

`PVideoCell` = `^TVideoCell`

Pointer type to `TVideoCell` ([1653](#))

`PVideoMode` = `^TVideoMode`

Pointer to `TVideoMode` ([1653](#)) record.

```
Tencoding = (cp437, cp850, cp852, cp866, koi8r, iso01, iso02, iso03, iso04,
             iso05, iso06, iso07, iso08, iso09, iso10, iso13, iso14, iso15, utf8)
```

Table 43.1: Enumeration values for type `Tencoding`

Value	Explanation
cp437	Codepage 437
cp850	Codepage 850
cp852	Codepage 852
cp866	Codepage 866
iso01	ISO 8859-1
iso02	ISO 8859-2
iso03	ISO 8859-3
iso04	ISO 8859-4
iso05	ISO 8859-5
iso06	ISO 8859-6
iso07	ISO 8859-7
iso08	ISO 8859-8
iso09	ISO 8859-9
iso10	ISO 8859-10
iso13	ISO 8859-13
iso14	ISO 8859-14
iso15	ISO 8859-15
koi8r	KOI8-R codepage
utf8	UTF-8 encoding

This type is available under Unix-like operating systems only.

```
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 43.2: Enumeration values for type TErrorHandlerReturnValue

Value	Explanation
errAbort	abort and return error code
errContinue	abort without returning an errorcode.
errRetry	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.

```
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 ([1663](#)) call.

An explanation of all fields can be found there.

```
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.

```
TVideoModeSelector = function(const VideoMode: TVideoMode;
                               Params: LongInt) : Boolean
```

Video mode selection callback prototype.

43.4.3 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 `UpdateScreen` (1664) 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` (1664) will write the actual characters to the screen.

```
VideoBufSize : LongInt
```

Current size of the video buffer pointed to by `VideoBuf` (1654)

43.5 Procedures and functions

43.5.1 ClearScreen

Synopsis: Clear the video screen.

```
Declaration: procedure ClearScreen
```

Visibility: default

Description: `ClearScreen` clears the entire screen, and calls `UpdateScreen` (1664) 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 \(1661\)](#), [UpdateScreen \(1664\)](#)

Listing: ./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.

43.5.2 DefaultErrorHandler

Synopsis: Default error handling routine.

[illegible]

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.

43.5.3 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` ([1661](#))

43.5.4 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` ([1657](#)), `GetVideoDriver` ([1659](#))

Listing: `./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.

```

43.5.5 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:

crHidden Hide cursor
crUnderLine Underline cursor
crBlock Block cursor
crHalfBlock Half block cursor

Note that not all drivers support all types of cursors.

Errors: None.

See also: `SetCursorType` ([1663](#)), `GetCapabilities` ([1656](#))

Listing: `./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.

```

43.5.6 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` (1664) will actually update the screen.

Errors: None.

See also: `LockScreenUpdate` (1661), `UnlockScreenUpdate` (1664), `UpdateScreen` (1664)

Listing: `./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;
    I:=0;
    While GetLockScreenCount<>0 do
        begin
            Inc(I);
            Str(I,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.

43.5.7 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` (1663) call.

Errors: None.

See also: `SetVideoDriver` (1663)

43.5.8 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` (1664), `GetVideoModeData` (1661)

Listing: `./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.

43.5.9 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` (1661) function to retrieve data for the supported video modes.

Errors: None.

See also: `GetVideoModeData` (1661), `GetVideoMode` (1659)

Listing: `./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.

```

43.5.10 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` (1660).

Errors: In case `Index` has a wrong value, `False` is returned.

See also: `GetVideoModeCount` (1660), `SetVideoMode` (1664), `GetVideoMode` (1659)

43.5.11 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` (1656)

43.5.12 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` (1664) 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 (1664) after the drawing, followed by a UpdateScreen (1664) call, all writing will be shown on screen at once.

For an example, see GetLockScreenCount (1658).

Errors: None.

See also: UpdateScreen (1664), UnlockScreenUpdate (1664), GetLockScreenCount (1658)

43.5.13 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 (1663)

Listing: ./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.

```

43.5.14 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` ([1662](#))

43.5.15 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` ([1661](#))) or after it was deactivated (i.e after a call to `DoneVideo`).

For more information about installing a videodriver, see `viddriver` ([1644](#)).

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` ([1644](#))

43.5.16 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 succesful, 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 succesfully, `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 succesful.

The mode can be set when the video driver has not yet been initialized (i.e. before `InitVideo` (1661) 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` (1660) and `GetVideoModeData` (1661) functions. To retrieve the current video mode, use the `GetVideoMode` (1659) procedure.

Errors: If the specified mode cannot be set, then `errVioNoSuchMode` may be set in `ErrorCode`

See also: `GetVideoModeCount` (1660), `GetVideoModeData` (1661), `GetVideoMode` (1659)

43.5.17 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` (1664) 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` (1661) is matched by exactly one call to `UnlockScreenUpdate`

For an example, see `GetLockScreenCount` (1658).

Errors: None.

See also: `LockScreenUpdate` (1661), `GetLockScreenCount` (1658), `UpdateScreen` (1664)

43.5.18 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` ([1655](#))

Chapter 44

Reference for unit 'wincrt'

44.1 Overview

The `wincrt` unit provides some auxiliary routines for use with the `graph` (643) unit, namely keyboard support. It has no connection with the `crt` (430) unit, nor with the Turbo-Pascal for Windows `WinCrt` unit. As such, it should not be used by end users. Refer to the `crt` (430) unit instead.

44.2 Constants, types and variables

44.2.1 Variables

`directvideo` : `Boolean`

On windows, this variable is ignored.

`lastmode` : `Word`

Is supposed to contain the last used video mode, but is actually unused.

44.3 Procedures and functions

44.3.1 `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` (1667), `nosound` (1667)

44.3.2 `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` ([1667](#))

44.3.3 nosound

Synopsis: Stop the speaker

Declaration: `procedure nosound`

Visibility: `default`

Description: `NoSound` does nothing, windows does not support this.

See also: `sound` ([1667](#))

44.3.4 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` ([1666](#))

44.3.5 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` ([1667](#))

44.3.6 textmode

Synopsis: Set indicated text mode

Declaration: `procedure textmode(mode: Integer)`

Visibility: `default`

Description: `TextMode` does nothing.

Chapter 45

Reference for unit 'x86'

45.1 Used units

Table 45.1: Used units by unit 'x86'

Name	Page
BaseUnix	100
System	1118

45.2 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.

45.3 Procedures and functions

45.3.1 fplOperm

Synopsis: Set permission on IO ports

Declaration: `function fplOperm(From: Cardinal; Num: Cardinal; Value: cint) : cint`

Visibility: default

Description: `FplOperm` 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` ([151](#)), the permissions are reset. When doing a `FpExecVE` ([148](#)) they are kept.

Errors: Extended error information can be retrieved with `FpGetErrno` ([154](#))

45.3.2 `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.

45.3.3 `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` ([1668](#)) call.

Errors: In case of an error (not enough permissions read this port), runtime 216 (*Access Violation*) will occur.

See also: `FpIOPerm` ([1668](#)), `ReadPortB` ([1669](#)), `ReadPortW` ([1670](#)), `ReadPortL` ([1670](#)), `WritePort` ([1670](#)), `WritePortB` ([1671](#)), `WritePortL` ([1671](#)), `WritePortW` ([1671](#))

45.3.4 `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` ([1668](#)) call.

Errors: In case of an error (not enough permissions read this port), runtime 216 (*Access Violation*) will occur.

See also: `FpIOPerm` ([1668](#)), `ReadPort` ([1669](#)), `ReadPortW` ([1670](#)), `ReadPortL` ([1670](#)), `WritePort` ([1670](#)), `WritePortB` ([1671](#)), `WritePortL` ([1671](#)), `WritePortW` ([1671](#))

45.3.5 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` (1668) call.

Errors: In case of an error (not enough permissions read this port), runtime 216 (*Access Violation*) will occur.

See also: `FpIOPerm` (1668), `ReadPort` (1669), `ReadPortW` (1670), `ReadPortB` (1669), `WritePort` (1670), `WritePortB` (1671), `WritePortL` (1671), `WritePortW` (1671)

45.3.6 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` (1668) call.

Errors: In case of an error (not enough permissions read this port), runtime 216 (*Access Violation*) will occur.

See also: `FpIOPerm` (1668), `ReadPort` (1669), `ReadPortB` (1669), `ReadPortL` (1670), `WritePort` (1670), `WritePortB` (1671), `WritePortL` (1671), `WritePortW` (1671)

45.3.7 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` (1668) call.

Errors: In case of an error (not enough permissions to write to this port), runtime 216 (*Access Violation*) will occur.

See also: [FpIOPerm \(1668\)](#), [WritePortB \(1671\)](#), [WritePortL \(1671\)](#), [WritePortW \(1671\)](#), [ReadPortB \(1669\)](#), [ReadPortL \(1670\)](#), [ReadPortW \(1670\)](#)

45.3.8 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 \(1668\)](#) call.

Errors: In case of an error (not enough permissions to write to this port), runtime 216 (*Access Violation*) will occur.

See also: [FpIOPerm \(1668\)](#), [WritePort \(1670\)](#), [WritePortL \(1671\)](#), [WritePortW \(1671\)](#), [ReadPortB \(1669\)](#), [ReadPortL \(1670\)](#), [ReadPortW \(1670\)](#)

45.3.9 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 `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 \(1668\)](#) call.

Errors: In case of an error (not enough permissions to write to this port), runtime 216 (*Access Violation*) will occur.

See also: [FpIOPerm \(1668\)](#), [WritePort \(1670\)](#), [WritePortB \(1671\)](#), [WritePortW \(1671\)](#), [ReadPortB \(1669\)](#), [ReadPortL \(1670\)](#), [ReadPortW \(1670\)](#)

45.3.10 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` (1668) call.

Errors: In case of an error (not enough permissions to write to this port), runtime 216 (*Access Violation*) will occur.

See also: `FpIOPerm` (1668), `WritePort` (1670), `WritePortL` (1671), `WritePortB` (1671), `ReadPortB` (1669), `ReadPortL` (1670), `ReadPortW` (1670)